Requirements Controlled Design: A Method for Discovery of Discontinuous System Boundaries in the Requirements Hyperspace

A Thesis
Presented to
The Academic Faculty

by

Peter Michael Hollingsworth

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Aerospace Engineering
Georgia Institute of Technology
March 2004

Requirements Controlled Design: A Method for Discovery of Discontinuous System Boundaries in the Requirements Hyperspace

Approved by:

Dr. D. N. Mavris, Adviser

Dr. S. Goodman

Dr. D. Schrage

Mr. C. Nickol

Dr. J. Craig

Date Approved: 9 April 2004

ACKNOWLEDGEMENTS

I would like to thank my adviser, Dr. Dimitri Mavris and my committee, Drs. Daniel Schrage and James Craig of the School of Aerospace Engineering, Dr. Seymour Goodman of the College of Computing & Sam Nunn School of International Affairs at the Georgia Institute of Technology, and Mr. Craig Nickol of NASA Langley Research Center for there assistance and guidance in completing this thesis and dissertation. Each provided a unique point of reference that helped me to improve the presentation of material and general readability. Additionally, I would like to thank Dr. Bryce Roth, Dani Soban, and Elena Garcia for there assistance over the course of this project, and Mr. Brian German for his assistance in assuring that the some of the more esoteric concepts contained within this thesis are understandable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

API	Application Programming Interface.
ASDL	Aerospace Systems Design Laboratory.
ATE	Advanced Turbine Engine.
c	Control Variable.
CE	Concurrent Engineering.
CO	Collaborative Optimization.
COAX	Coaxial Rotor.
CPU	Central Processing Unit.
CV	Control Variable.
EA	Evolutionary Algorithm.
EELV	Evolved, Expendable Launch Vehicle.
F	Universal Unfolding.
F_c	c Partial Functions of F.
FPI	Fast Probability Integration.
$\frac{T}{W}$	Thrust to Weight Ratio.
$\frac{W}{S}$	Wing Loading.
GA	Genetic Algorithm.
GP	Gaussian Process.
GW	Gross Weight.
H	Potential Equation, Performance Criteria.
H_2	Hydrogen.
HOGE	Hover Out of Ground Effect.
HP	Horsepower.
IO	Input & Output.
IPD	Integrated Product Development.

IPPD Integrated Product and Process Development.

I_{sp} Specific Impulse.

JDAM Joint, Direct Attack Munition.

JP Jet Propellant.

JPDM Joint Probabilistic Decision Making.

LHX Light Helicopter Experimental.

$\mathbf{C}$ Control Variable Vector.

$\mathbf{F}$ Free Variables, these are $\mathbf{Y}$s or non-optimized state variables, that are allowed to "float" freely in the input file.

$\mathbf{R}$ Real Space.

$\mathbf{R_f}$ Fuel Weight to Gross Weight Ratio.

$\mathbf{R}^l$ Control Variable Space.

$\mathbf{R}^n$ State Variable Space.

$\mathbf{T}$ Tracked Variables, these are $\mathbf{C}$s which are external objective function outputs.

$\mathbf{X}$ State Variable Vector.

MDO Multidisciplinary Design Optimization.

M_F Catastrophe surface.

MSPEA Modified Strength Pareto Evolutionary Algorithm.

N-S Navier-Stokes.

NSGA-II Non-dominating, Sorting Genetic Algorithm.

p Critical Point, Momentum.

P Pressure, Payload.

P_s Specific Excess Power.

PSE Problem Solving Environment.

q Quadratic Form.

R Range.

RCD Requirements Controlled Design.

RDS Robust Design Simulation.

RSE Response Surface Equation.

RSM Response Surface Methodology.

SA Simulated Annealing.

SMR Single Main Rotor.

SOA State of the Art.

SPEA Strength Pareto Evolutionary Algorithm.

SPEA2 Strength Pareto Evolutionary Algorithm 2.

SQP Sequential Quadratic Programming.

SV State Variable.

T_3 Combustor Inlet Temperature.

T_4 Combustor Exit Temperature.

TDM Tandem Rotor.

TIES Technology Identification, Evaluation, and Selection.

TIF Technology Impact Forecasting.

TLTR Tiltrotor.

UTE Unified Tradeoff Environment.

V Vector Space, Velocity.

VBA Visual Basic for Applications.

VLS Vertical Launch System.

VSLCD Virtual Stochastic Life-Cycle Design.

V/STOL Vertical/Short Take-Off and Landing.

W Weight.

x State Variable.

SUMMARY

The drive toward robust systems design, especially with respect to system affordability throughout the system life-cycle, has led to the development of several advanced design methods. While these methods have been extremely successful in satisfying the needs for which they have been developed, they inherently leave a critical area unaddressed. None of them fully considers the effect of requirements on the selection of solution systems. The goal of all of current modern design methodologies is to bring knowledge forward in the design process to the regions where more design freedom is available and design changes cost less. Therefore, it seems reasonable to consider the point in the design process where the greatest restrictions are placed on the final design, the point in which the system requirements are set.

Historically the requirements have been treated as something handed down from above. At best, a negotiation would take place to settle on a specific set of requirements that were acceptable to both the customer and solution provider. However, in both of these cases, neither the customer nor the solution provider completely understood all of the options that are available in the broader requirements space. If a method were developed that provided the ability to understand the full scope of the requirements space, it would allow for a better comparison of potential solution systems with respect to both the current and potential future requirements. Furthermore, by evaluating the inclusion of current or new technologies, it is possible to identify not only which systems can fulfill a certain set of requirements, but also which technologies will enable the satisfaction of those requirements.

The key to a requirements conscious method is to treat requirements differently

from the traditional approach. The method proposed herein is referred to as Requirements Controlled Design (RCD). By treating the requirements as a set of variables that *control the behavior* of the system, instead of variables that only *define the response* of the system, it is possible to determine a priori what portions of the requirements space that any given system is capable of satisfying. Additionally, it should be possible to identify which systems can satisfy a given set of requirements, which system is the best choice given knowledge of the future requirements trend, and the locations where a small change in one or more requirements poses a significant risk to a design program. This thesis puts forth the theory and methodology to enable RCD, and proposes and evaluates two basic methods, a grid search and an evolutionary, global optimization scheme, for finding the technology boundaries of a system in the requirements hyperspace. Finally a specific method using a Pareto seeking evolutionary algorithm to discover the location of predetermined technology boundaries is described and validated. The algorithm, called Modified Strength Pareto Evolutionary Algorithm (MSPEA), uses advancements in multi-objective, global optimization to enable finding the technology induced boundaries in both the design and requirements spaces. Using MSPEA an evaluation of the U.S. Army's Light Helicopter Experimental (LHX) program is also presented. This evaluation demonstrates the capability of both RCD and the MSPEA algorithm and validates the ability of the algorithms to find a historically feasible space.

CHAPTER I

INTRODUCTION & MOTIVATION

The goal of most modern aerospace design methodologies has been to bring knowledge forward in the design process to increase systems effectiveness and affordability. While these methods have been successful in fulfilling this goal, they have left a critical area unaddressed. None of them fully consider the effect of requirements on the selection of solution systems. Since modern methods desire to make decisions where the greatest amount of flexibility is available; it seems reasonable to consider the point in the design process where the greatest restrictions are placed on the final design, the point in which the requirements are set.

In the past requirements were often considered as only minorly varying if not absolute, with little allowable trade-off between the settings of one vs. another requirement. This led to designs that were inherently compromised by changing needs, preferences and desires. If a method were developed that provided a view of the entire requirements space, it would allow for a better comparison of potential solution systems with respect to both the current and potential future requirements. Furthermore, including both current and new technologies, it would be possible to identify not only which systems can fulfill a certain set of requirements, but also which technologies will enable the satisfaction of those requirements.

The key to this new method is to treat requirements differently from how they are dealt with currently. This new ideal is encompassed in method proposed and demonstrated herein, Requirements Controlled Design (RCD). By treating the requirements as a set of variables that *control the behavior* of the system, instead of variables that only *define the response* of the system, one can identify where in the requirements

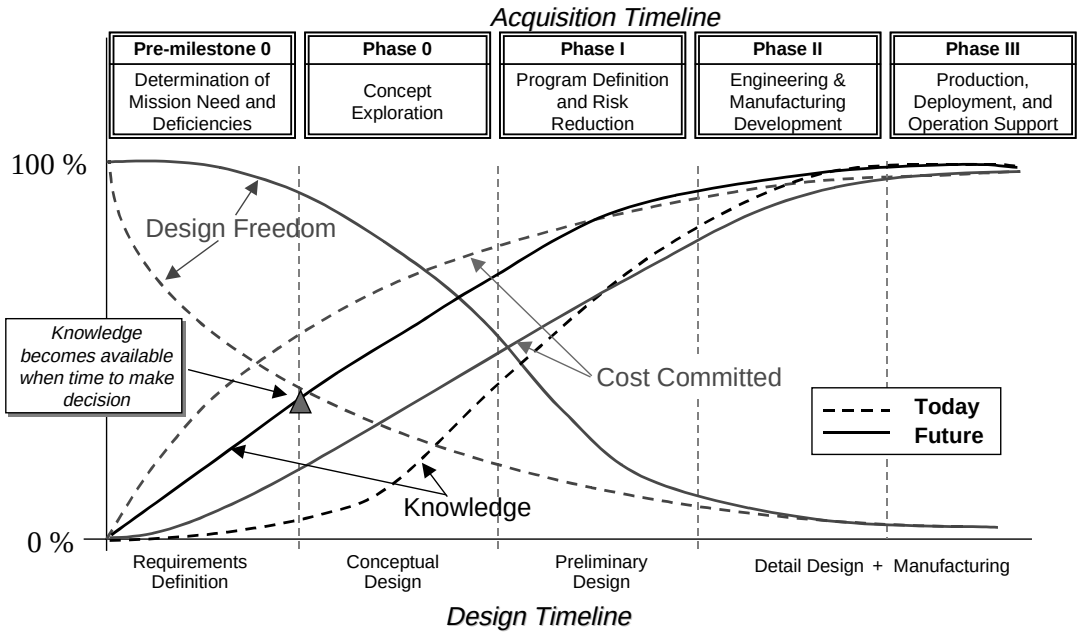


Figure 1: Design Freedom, Knowledge, Cost Relationship [1]

(hyper)space a particular system lies. Additionally, one can determine which systems are available at any point, which system is the best choice given knowledge of the future requirements trend, and the locations where a small change in one or more requirements poses a significant risk to a design program.

1.1 Motivation

Any complex system can be defined by a conglomeration of component, subsystem, and system attributes. The set of all of these attributes functionally defines a specific system. This is as true of aerospace systems as it is for any other complex system. Experience teaches us that it is possible to loosely determine the system type with far fewer variables than those necessary to determine the final vehicle. For example, by choosing to build a commercial jet transport, the designer or manufacturer has already eliminated most system types. This reduction in "design freedom" is consistent with the ideas shown in Figure 1.

Further, because requirements may change over the course of a program, both

during development and the remainder of the system's life-cycle, a lack of thorough understanding of the effect of the requirements can easily produce systems that perform poorly. This was cited by Mavris and DeLaurentis.

> The process of system engineering has always emphasized the definition of requirements as the first step toward product development. Typically, however, these requirements were examined in isolation from the potential systems and technologies they would likely impact. Further, requirements during design were treated deterministically, which sometimes led to non-robust and poor performing actual systems which encountered different requirements. Thus, there is a need to examine requirements early on and in a new way.... [1]

This understanding led to the development of the Unified Tradeoff Environment (UTE) [2, 3, 4, 5], which is designed to investigate the "simultaneous impact of requirements, product design variables, and emerging technologies during the concept formulation and development stages.... [1]." The UTE investigates the effect that changes in the requirements, system attributes, and technologies have on a baseline system, i.e. perturbations about a point. Furthermore, the UTE requires that an investigation into the effects of varying requirements is performed on an otherwise "frozen" system [3, 4, 5] . While both of these limitations reduce the complexity of a combined design, requirements, and technology space investigation, they also limit the applicability of the method.

1.1.1 Stochastic Requirements

A basic premise of methods such as the UTE is that requirements do not remain constant over the life-cycle of a system. In most cases these changes are small and produce only minimal deviations in the characteristics of the final system. These are the types of requirements changes that the UTE was designed to investigate. However,

there are two types of changes in the values of requirements that significantly affect the final system.

1. **Significant change in a given requirement**: This places the final design in a completely different region of the requirements (hyper)space.

2. **Changing requirements cross a technological or physics based boundary**: This produces a total discontinuous change in the final system.

Either of these events can easily upset the assumptions made when working in the UTE, which is based upon perturbations about a baseline point. The first can typically be avoided by careful consideration of end user needs ahead of time, and is, therefore, not of great interest to the remainder of this dissertation. The second event, since it requires a true simultaneous investigation of the requirements, vehicle attributes, and current or future technologies, presents a much greater challenge. Using a gas-dynamics analogy, to understand the second type of event, one must look at design as an equilibrium or even a nonequilibrium process.

1.1.2 Effect of System Requirements on the System

As mentioned at the beginning of this chapter, if an engineer were asked to design a commercial transport that can travel 4,000 nm in twelve hours and carry 250 people, there is little doubt as to what the subsystems' characteristics would be that comprise this notional transport:

- Tubular fuselage
- Swept, high aspect ratio wing
- High-bypass ratio, turbine engines
- etc.

If the flight time requirement were lowered a small amount to eleven hours, the final configuration would probably change very little. This is the type of change that the UTE and other methods were developed to handle. The "frozen" assumption holds, as the perturbation in the requirement is only slightly off of the initial "equilibrium." However, if the time required drops to two and one-half hours, the configuration would be expected to change significantly. Additionally, the subsystems that make up the final system would change. Therefore, the original assumptions no longer hold. The perturbation about a baseline point is no longer a valid approach. The design has to be based off a new equilibrium point. The inability to study such problems with a "frozen" design was documented by the author during an investigation into the effect of the system level requirements on a hypersonic strike fighter [6].

1.1.2.1 Potential Solutions in a Requirements Hyperspace

The example above is only one, albeit an extreme one, of the many changes in requirements that can happen to any aerospace system during its design and service life. Taking a step back and investigating the whole of the available requirements (hyper)space, i.e. no specific values of requirements are set, only the types of requirements are chosen, an engineer can easily envision that several systems or system types may occupy the space. A notional two-dimensional representation of this is given in Figure 2.

Imagine a baseline system of system type B in Figure 2. If an engineer performed perturbations about that baseline point, with respect to requirements 1 & 2, as shown in Figure 3, he or she could easily achieve a different system of type B; however, all of the other system types would be inaccessible using a "frozen" method such as the UTE.

While the existence of multiple solutions is easy to visualize in two-dimensions,

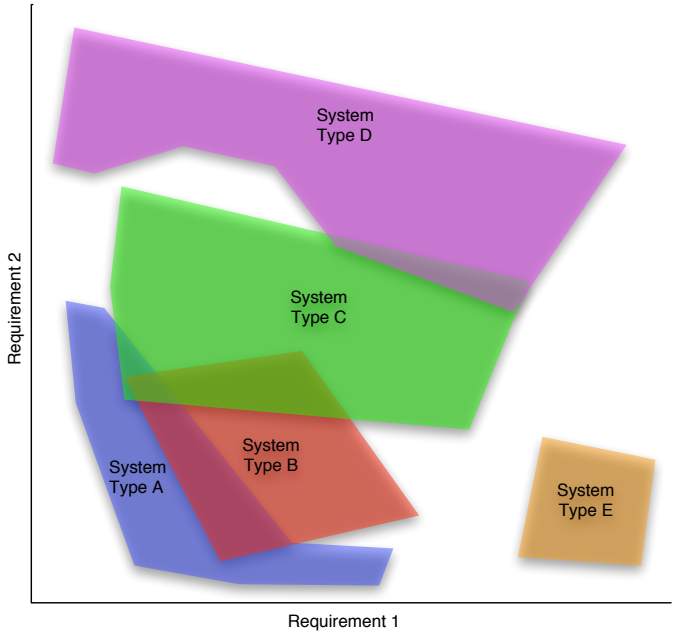


Figure 2: Notional Two-dimensional Requirements Space

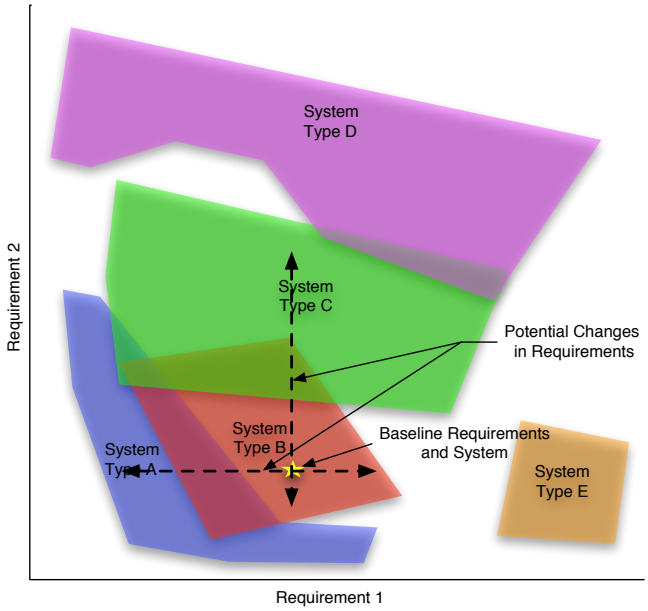


Figure 3: Notional Perturbation of Initial Requirements

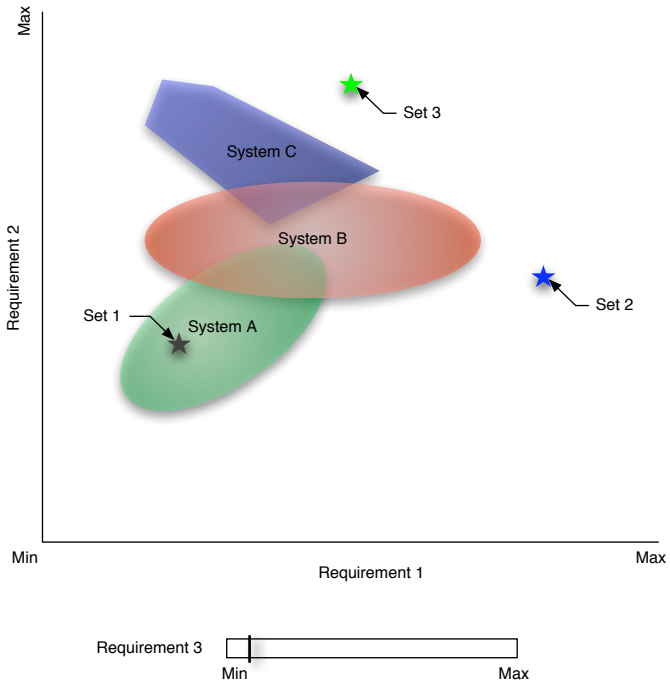


Figure 4: Notional Three-Dimensional Requirements Space, Requirement 3 Level 1

it is valid for any n-dimensions. Figures 4, 5, 6, 7, and 8 illustrate this for three-dimensions, by depicting five slices taken at different values of Requirement 3. Additionally, there are three combinations (sets) of Requirements 1 & 2 shown on the graph. In Figure 4, Requirements Set 1, is within the available solution space for system A. Figure 5, which shows a higher setting for Requirement 3, again has Set 1 in the solution space for system A. Additionally, System D now shows-up in the requirements space. By Figure 6, system B has disappeared while a new system, E has also appeared, and none of the requirements sets are within an available solution space. In Figure 7, all of the potential systems except E have disappeared. Additionally, requirements set 2 is now within the solution space for system E. By Figure 8, Requirement 3 has become so stringent that no potential solutions are available. Also requirements set 3 could not be satisfied for any value of Requirement 3. The investigation of this type of scenario requires a capability beyond that provided

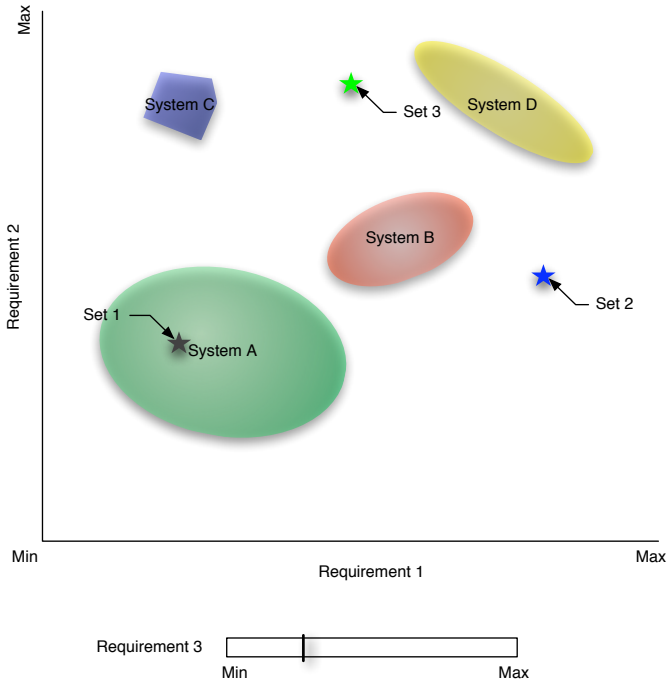


Figure 5: Notional Three-Dimensional Requirements Space, Requirement 3 Level 2

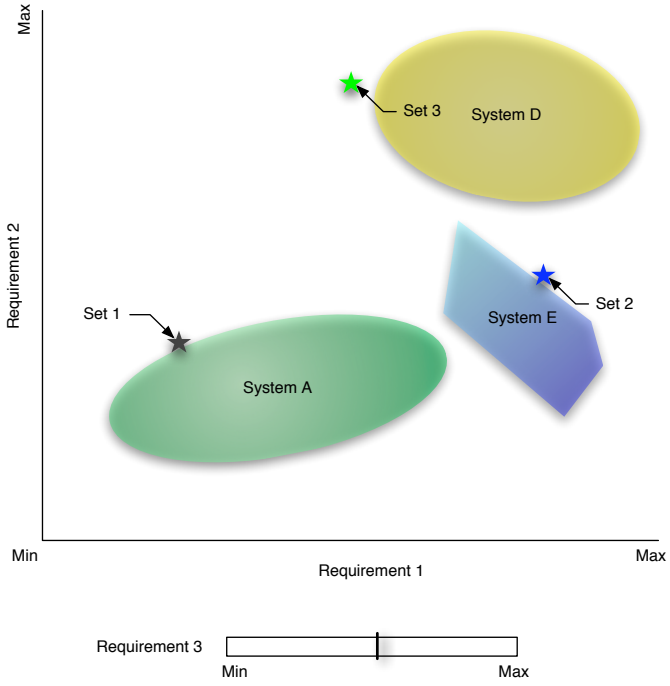


Figure 6: Notional Three-Dimensional Requirements Space, Requirement 3 Level 3

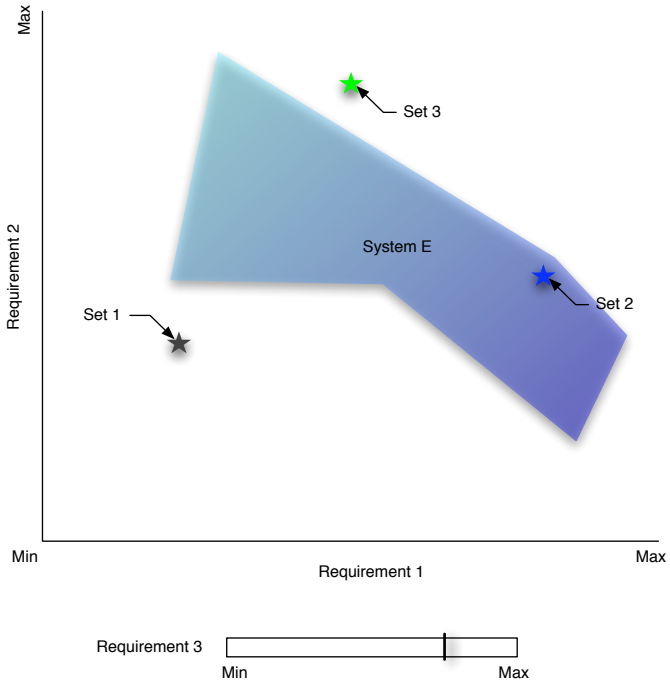


Figure 7: Notional Three-Dimensional Requirements Space, Requirement 3 Level 4

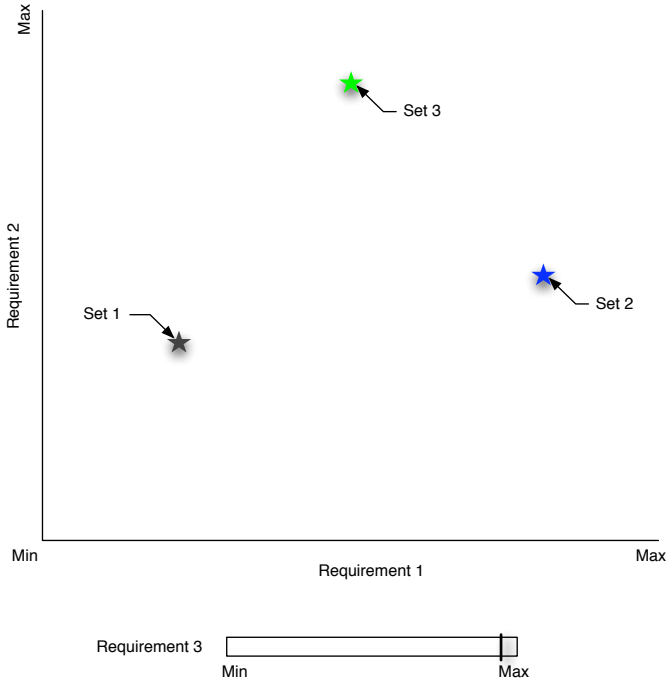


Figure 8: Notional Three-Dimensional Requirements Space, Requirement 3 Level 5

by the UTE. Another aspect of the notional requirements (hyper)space, shown in both Figures 2 & 3, is that more than one system or system type may exist for a given value of the requirements.

1.1.2.2 Nonunique Solutions to Requirements

The existence of multiple possible solution system types, i.e. nonunique solutions, to a particular requirement, is not just a notional idea. It was demonstrated by the author in a study of high-speed strike systems [7]. In that study the system types that occupied the same portions of the requirements space were a hypersonic strike fighter, and a hypersonic cruise missile. Using the notion of multiple potential solutions to consider a modern tactical weapons system, the system types that may co-exist within the requirements space include:

- Manned aircraft with an unpowered weapon
- Unmanned aircraft with an unpowered weapon
- Guided standoff weapon
- Remote platform mounted laser

The Mach number requirements space for modern propulsion systems, shown in Figure 9, is similar.

Using Figure 9 as a starting point it is possible to quickly determine which propulsion system types are available for a given Mach number. These are shown in Figure 10. Figure 10(a), shows the systems available at Mach 0.8. These include the turbofan, turbojet, and rocket. In Figure 10(b), which shows Mach 2.5, the turbofan is no longer an efficient option, and the hydrocarbon and hydrogen ramjet are now available as choices. At Mach 4.0, shown in Figure 10(c), the turbojet is no longer available and the scramjet appears as an option, though at the bottom of its efficient Mach number range. In addition to the possibility that multiple system types exist

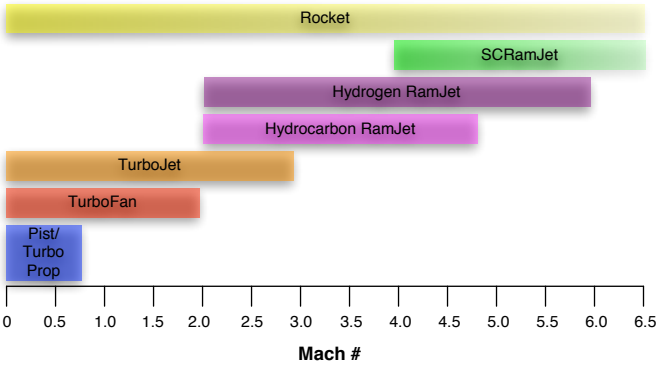
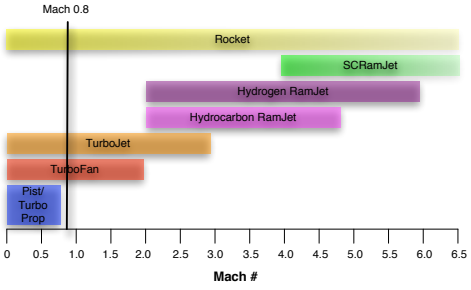
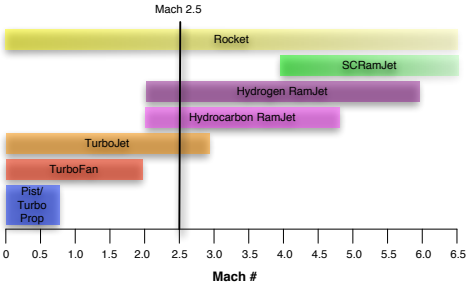


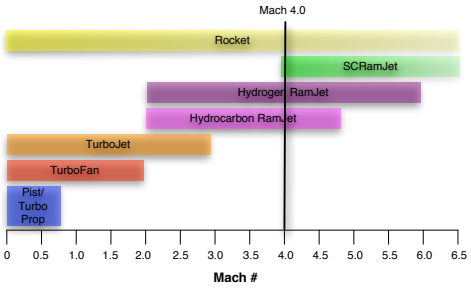
Figure 9: One-dimensional Propulsion System Requirements Space



(a) Mach 0.8



(b) Mach 2.5



(c) Mach 4.0

Figure 10: Propulsion Systems Available at Specific Mach Numbers

for a given set of requirements; it is also possible that multiple solution systems exist within a given system type.

Historically, multiple solutions have been proposed for a given set of requirements, e.g. the Delta IV and Atlas V for the US Air Force's EELV requirement or the 777-300ER and A340-600 in a specific segment of the commercial aircraft market. The number of varieties of systems for commercial or other civil aircraft design is not all that great. However, unlike fixed-wing design, which coalesced on stressed skin monoplanes for most systems in the 1930s and 1940s, rotorcraft applications have always had a diverse selection of systems for similar sets of requirements. Part of this is due to the more diverse types of requirements that a civil or military rotorcraft is required to fulfill. Examples of some of the diverse rotorcraft systems are given in Figure 11.

The same, multi-system, trend is also present in missile design, i.e. ballistic and cruise missiles, and small aircraft, i.e. piston and turboprop/shaft. The trend is not limited to aerospace systems, for example there are different hull types available that fill a range of requirements in the field of naval architecture. Combined with the inability to distinguish between the multiple available solution systems and system types; "frozen" design architectures such as the UTE cannot handle truly "revolutionary" technologies. A prime example of this is the introduction of the jet aircraft.

1.1.2.3 Historical Example of a Radical Technology Leap

During the 1930's the speeds attainable by aircraft began to enter the region where compressible effects become important. Initially these effects were most pronounced at the tip of the propeller. During the Second World War, the need for faster, higher flying aircraft, quickly reached the reasonable physical limits of propeller efficiencies at high speeds. The great need for aircraft that could fly higher and faster than possible using piston-prop technology propelled development the development of the

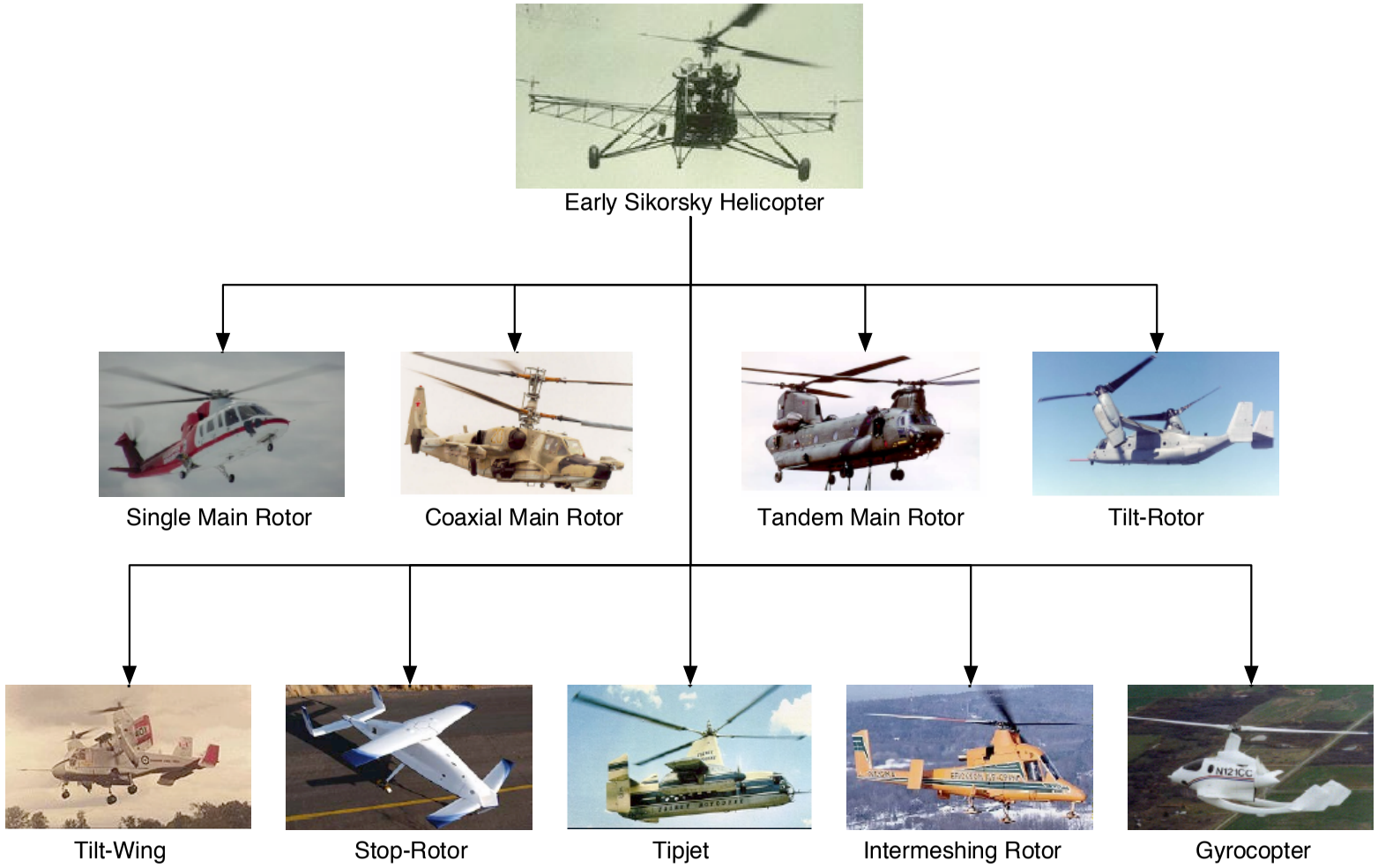


Figure 11: Example of Potential Rotorcraft Solution Systems

jet aircraft, in spite of its relative lack of maturity. Years later, jet powered aircraft are extremely common, well developed, and have enabled whole new aircraft applications. Had it not been for the primacy of the requirement to fly higher and faster, the development of the jet engine would most probably been delayed if not totally shelved.

1.1.3 Need for a More Capable Methodology to Investigate the Effect of System Requirements

The problem of investigating radically new or different technologies, necessitates an approach which is not wedded to a baseline system/vehicle. Furthermore, it requires that all of the important variables: the requirements, system attributes, and technologies, be considered in consort. The current methods are limited by the constraint that the configuration and system requirements are held constant whenever the engineer varies the technologies, through the use of subsystem metric multipliers. Effectively the engineer is measuring the sensitivity, the partial derivative, of the system response to the change in technology. The same holds true for changes in requirements or system attributes. What is needed to properly study the effect of requirements is a method that is not limited by the need to "freeze" the design. For this reason it is necessary to look to a branch of mathematics for guidance: bifurcation, catastrophe, and chaos theories.

1.1.4 Goals for the Improved Methodology

Any improvement in a design methodology should address the above problems. Specifically it needs to provide the designers & engineers and the decision makers with information that enables them to:

1. Determine which systems or system types are available for any given settings of requirements.

2. Assess the risk that uncertain requirements have upon the system or system type of choice.

3. Understand the "best" that can be achieved with a given level of technology. Specifically so that the decision makers can set the specific value of the requirements appropriately.

4. Identify the necessary areas of improvement, such that technology investments can be made. Either to improve an existing system or enable a new one.

5. Ultimately enable the decision makers to make schedule, performance, and cost trade-offs with respect to needs and technology developments.

While all of these goals are individually important; it is combined that they have the most effect. Combining the above attributes allows both the producer and consumer of the complex system to better understand each other's needs and capabilities. Ultimately, this will help lead to the production of systems that better meet the customers needs, are more robust, and are produced more efficiently.

CHAPTER II

BACKGROUND

The need to increase the understanding of the behavior of complex systems, improve affordability, and manage technology development, has led to the formulation of modern design techniques. This process is ongoing; however, many of the current methods neglect a basic fact, i.e. a large portion of the design freedom is inherently eliminated when choosing the system design requirements. Depending on how the requirements are set, the system type may actually be pre-specified, thereby eliminating most of the design freedom and locking in the costs inherent with that system. The necessity to understand this effect has led to the study of not only modern design methods, but also a branch of mathematics that deals with changing system behavior with respect to small changes in a set of key variables, Catastrophe Theory, materials science, and economics.

2.1 Modern Complex System Design Methods

The primary purpose of modern complex design methods is to bring forward knowledge in the design process, where there is more design freedom and lest cost associated with changes. This goals of this new paradigm are illustrated in Figure 12. To understand where the current state-of-the-art in complex system design methods stands, it is useful to understand the history of aerospace advanced design methodologies. Two early methods developed to meet this goal were Concurrent Engineering (CE) and Integrated Product Development (IPD). The expansion and combination of the ideas set forth in both CE and IPD to address, among other things, affordability produced Integrated Product and Process Development (IPPD) [8].

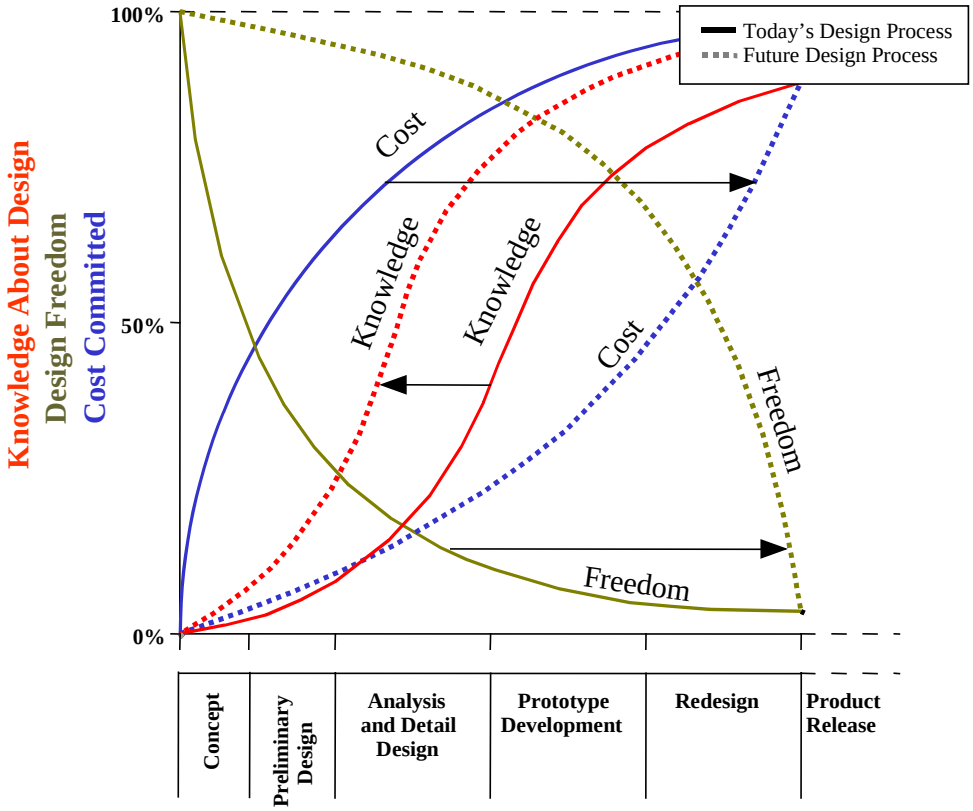


Figure 12: Cost, Knowledge, & Freedom Relations, Adapted[9, 10]

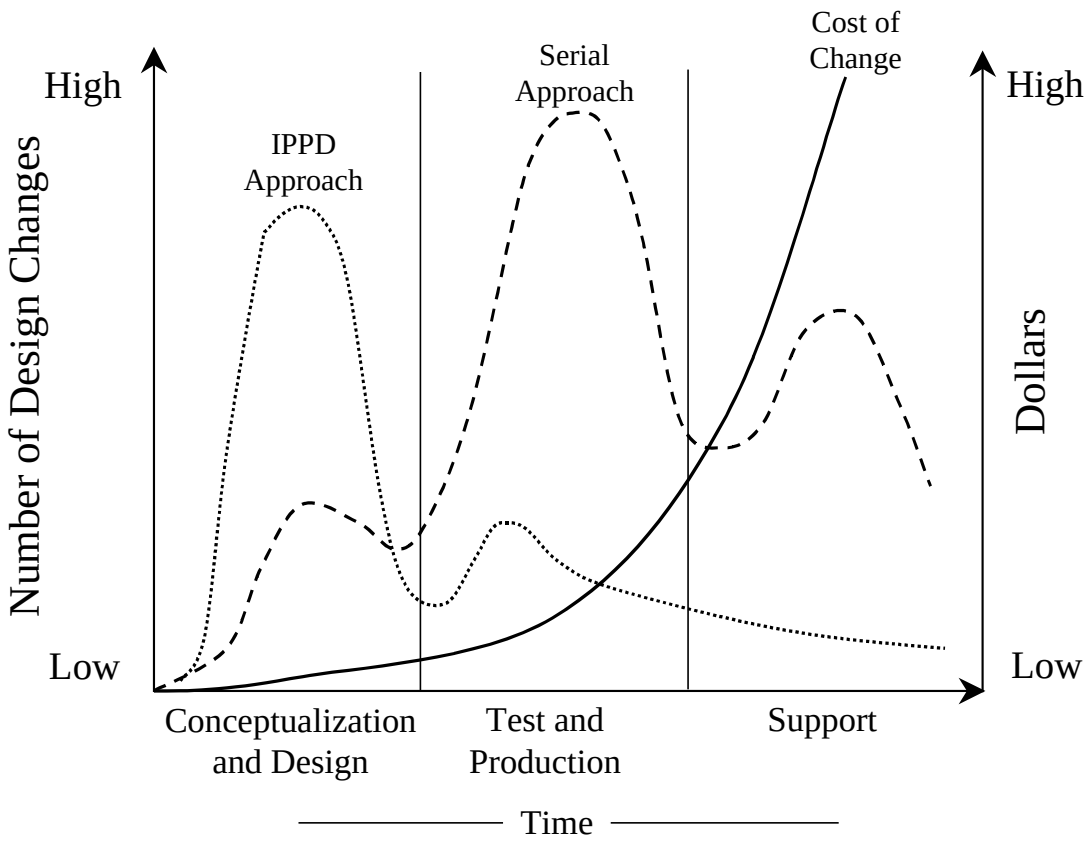


Figure 13: Comparison Between Serial and IPPD Design Approaches [12, 10]

2.1.1 Integrated Product and Process Development

Over the last several years there has been ongoing and increasing pressure to bring knowledge forward in the design process of complex systems. The use of IPPD techniques attempts to "bring forward" information in the design sequence, thereby allowing the most affordable design to be chosen, and the requisite changes made before costs are locked in [11]. The comparison between the IPPD and serial design approaches is illustrated in Figure 13.

The use of the IPPD method, in both its generic form and the form as modified by Georgia Tech, allows the engineer and program manager to decompose the product and process design trade iterations [13]. The generic IPPD process is shown in Figure 14, while the specific methodology developed at Georgia Tech is given in Figure 15.

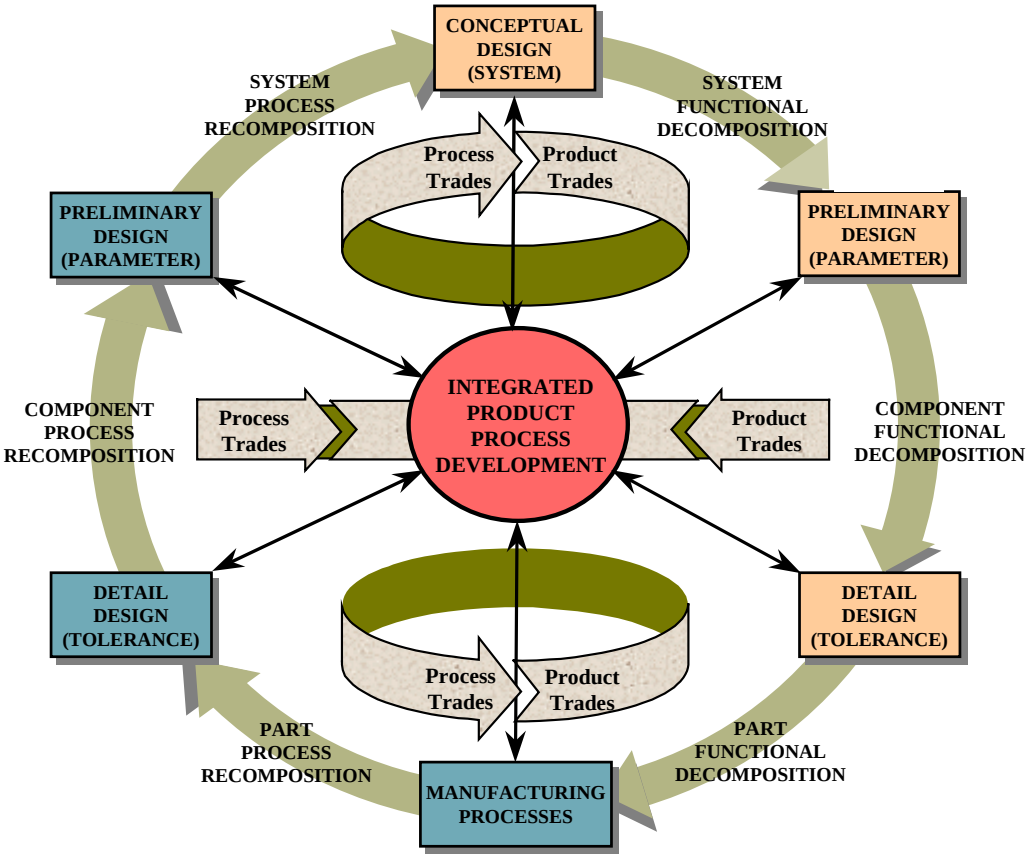


Figure 14: Hierarchical Process Flow for Large Scale System Integration [14, 10]

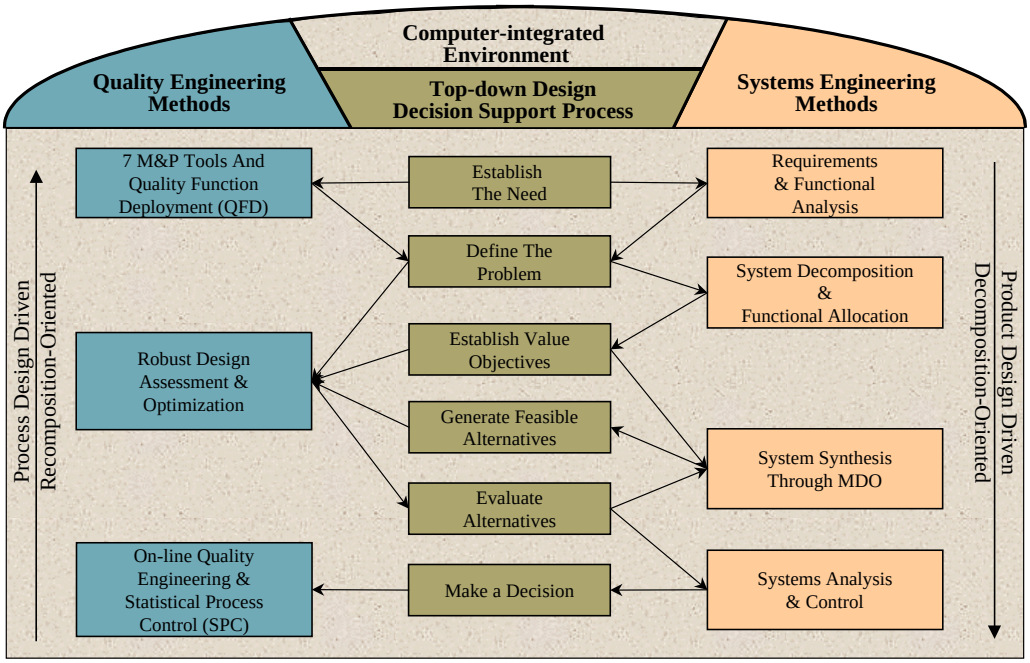


Figure 15: Georgia Tech IPPD Methodology [13, 10]

The implementation of the IPPD methodology allows the engineer to more readily investigate the effect of the uncertainty associated with the design, certification, manufacturing, and operational aspects of a complex aerospace systems life-cycle. This uncertainty, coupled with the unstable nature of many "optimum" designs, resulted in a desire to ensure robustness of the final system to uncertain factors throughout the system life-cycle. To this end the Aerospace Systems Design Laboratory (ASDL) developed Robust Design Simulation (RDS), a more specific and detailed IPPD methodology.

2.1.2 Robust Design Simulation

The initial RDS techniques were developed and implemented in ASDL in the early 1990s. The RDS methods allow the designer to identify "key product and process characteristics as well as their relative contributions to the chosen evaluation criterion in the presence of risk and uncertainty [15]." The key purpose of the RDS method

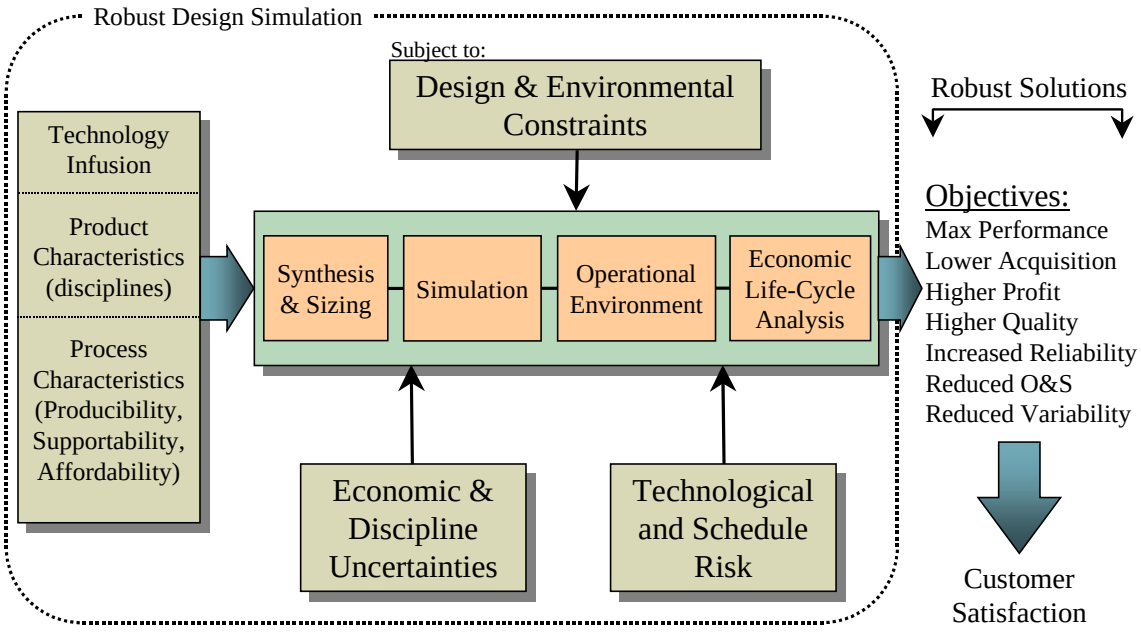


Figure 16: ASDL Robust Design Simulation [15, 10]

is to ensure that the final system will meet its goals and satisfy the customer. The RDS system is depicted in Figure 16.

The RDS structure allows the assessment of both the design variability and its feasibility & viability. A feasible design is one that is technologically possible, while a viable solution is one that meets the economic performance goals [15]. For the study of the system requirements space, a solution must be both feasible and viable to be considered successful. It is useful to point-out that in the case of RDS as it is currently implemented, the requirements, whether they are performance and/or economic in nature are considered as preset invariants. The ability to investigate a future system's capabilities with respect to both its physical and economic performance allows the designer to determine ahead of time, when a requirement or requirements will not be met. Additionally, if the design-space exploration is performed correctly, an assessment of the improvement in the offending performance metric that is required to achieve a feasible & viable design can be achieved. In order to implement the RDS

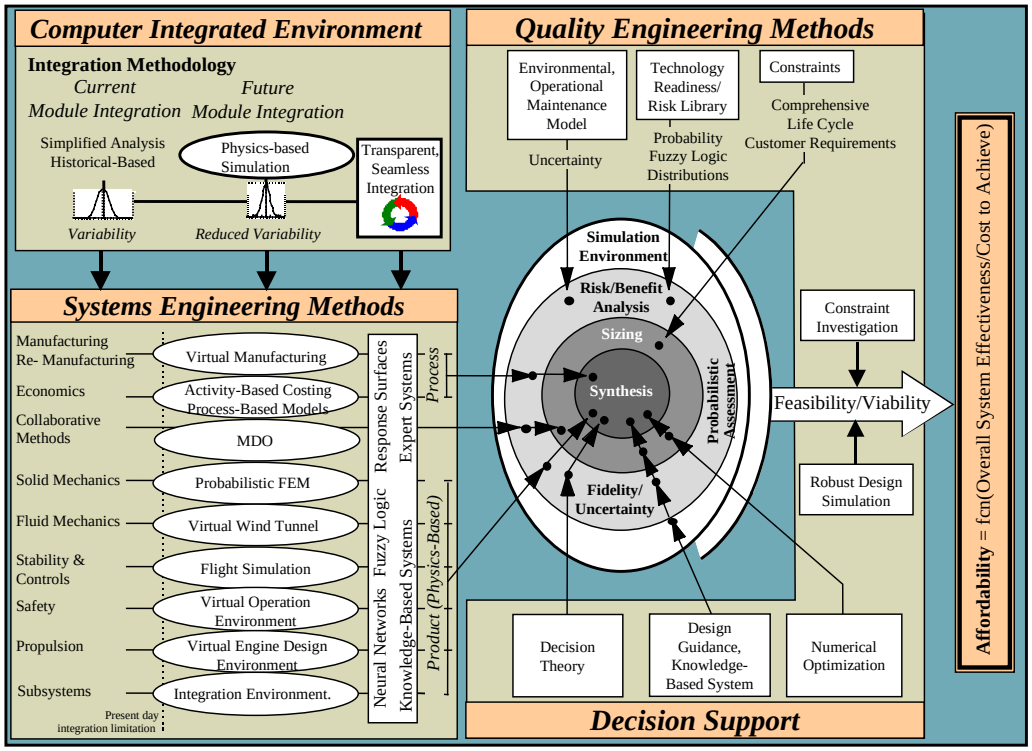


Figure 17: Virtual Stochastic Life Cycle Design Environment [16]

method, in a reasonable amount of time, it has been customary to replace the actual analysis and experimentation with meta-models in problems of high computational intensity or high combinatorial complexity.

2.1.3 Virtual Stochastic Life Cycle Design Environment

The Virtual Stochastic Life-Cycle Design (VSLCD), environment was developed to "facilitate decision making (at any level of organization) to reach affordable conclusions with adequate confidence [16, 17]." The VSLCD method is intended to encompass the entire life-cycle of a system from design to disposal. Additionally, the S in VSLCD indicates that the environment was designed to encompass the change in uncertainties over time [16]. VSLCD is intended to create an overarching environment that enables the cost, knowledge, and design freedom goals shown in Figure 12. A graphical representation of VSLCD is shown in Figure 17.

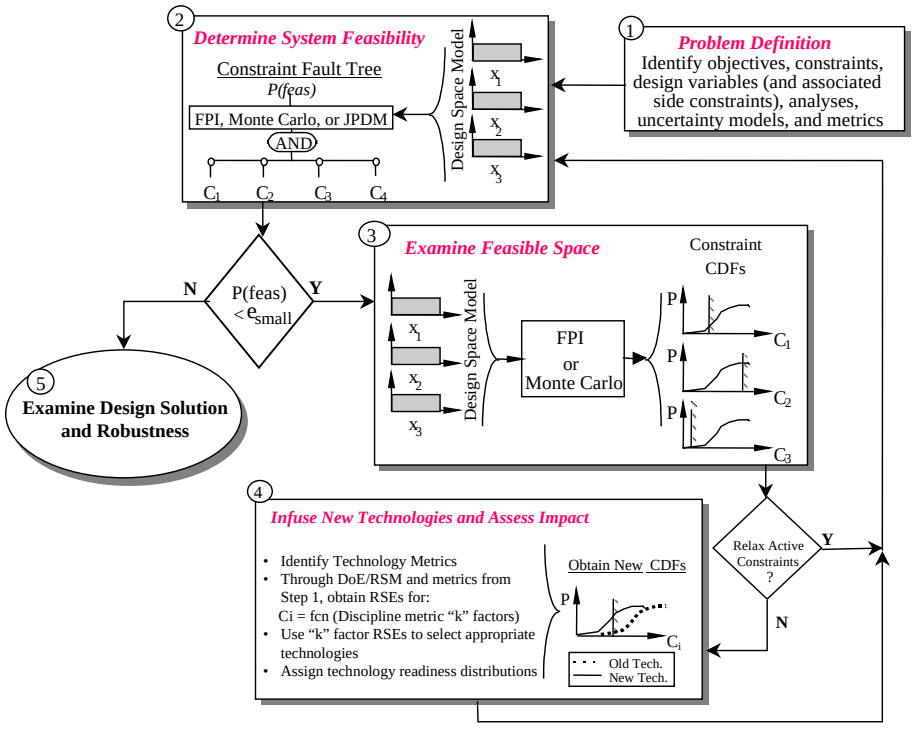


Figure 18: Conceptual Feasibility and Viability Method [18, 19, 20, 10]

VSLCD requires RDS, a method of investigating the feasibility and viability of a design or design space, and a means of evaluating and grading potential technologies, their benefits and risks. RDS was mentioned above, the means of evaluating system feasibility and viability and assessing technologies are further discussed below.

2.1.4 Design Space Exploration, System Feasibility & Viability Evaluation

Design space exploration, is a means of determining what percentage and portions of a range of specific systems are capable of meeting the mission performance, regulatory, and affordability requirements. The basic steps of this process are shown in Figure 18. Note that a means of evaluating the effect and applicability of technologies is critical to identify what portions of the design space are feasible and viable. How to perform this identification will be discussed later.

Critical to the implementation of such a method are system models and probabilistic methods such as Monte Carlo, Fast Probability Integration (FPI) [21], or Joint Probabilistic Decision Making (JPDM) [22]. Monte-Carlo is the most commonly implemented scheme, though the FPI method is gaining usage. The reason for this is that the FPI typically requires far fewer data points and is run directly on the analysis tools being used. However, because of the computational requirements to perform a Monte Carlo analysis it is common to replace the analysis tools with meta-models.

2.1.5 Meta-modeling

The use of meta-models in place of the system analysis tools is performed primarily to reduce the computational requirements of performing a design space exploration. A meta-model is defined as:

Definition 1. An approximation of a complex analysis model is called a **meta-model** [23].

Put succinctly a meta-model is a lower-order approximation of a model of reality. Much as many people would like to believe otherwise, the Navier-Stokes (N-S) fluid-dynamics equations are really only an approximate model, though a highly accurate one, of the actual electromagnetic interactions between atoms and molecules that define the behavior of fluids. The N-S equations, however, are relatively difficult to solve, even numerically. Therefore in hopes of minimizing the time required to investigate a phenomenon an engineer may decide to replace the N-S equations with a simpler model, i.e. Euler Equations, Full Potential, or even Linear Potential. In each of these cases, the engineer is forsaking fidelity for the purpose of decreasing the complexity and time associated with finding an answer. In the above cases the simplifications can be arrived at, mathematically, from the original N-S equations. However, when dealing with meta-modeling, there is typically no mathematical path from the original model to the approximation.

2.1.5.1 Meta-modeling Methods

Many methods of meta-modeling have been investigated and implemented in complex systems design. Each one of them has its inherent advantages, disadvantages, and assumptions. Some of the more common meta-models are listed below.

- **Response Surface Equations**: The Response Surface Equation (RSE), the fundamental functional representation of the meta-models developed using the Response Surface Methodology (RSM), is generally a linear statistical regression function, i.e, the coefficients in the equation are constant. The typical form of the RSE is quadratic with linear interactions, this form is given in Equation 1.

$$R = b_0 + \sum_{i=1}^{n} b_i x_i + \sum_{i=1}^{n} \sum_{j=i}^{n} b_{ij} x_i x_j \tag{1}$$

  Additional forms of the RSE are available, including linear, cubic, quartic, etc. Further, it is possible to perform variable transformations on the parameters and responses to improve the fit of the meta-model. The benefits of using RSM and RSEs is that creation and operation of the meta-model is relatively quick and simple. Further because the parameters are contained, explicitly in the equation the operation of the RSE is highly transparent. Unfortunately, there exists a class of problems, generally those of high complexity and non-linearity for which RSEs are unsuitable.

- **Kriging**: Kriging is an empirically developed nonlinear meta-modeling technique. Typically Kriging is composed of a nonlinear global extension to a local linear meta-model [24, 25, 26, 27]. This allows a significant increase in the meta-modeling capability while preserving a large amount of the transparency typically associated with linear meta-models, such as RSEs. Because of their combined linear/nonlinear nature Kriging models may not be appropriate for all models.

- **Neural Networks**: Neural networks are a nonlinear, Bayesian modeling technique consisting of artificial "neurons" which are trained to give a particular response to the input parameters [25, 28, 29, 30, 31]. Neural networks are capable of representing highly nonlinear spaces, including those that are almost discontinuous. Furthermore, because they are Bayesian in nature it is possible to improve the performance of the meta-model by adding new data over time. Unfortunately, there is a significant initial computational expense associated with neural networks, particularly in the training process.

- **Gaussian Processes**: The Gaussian Process (GP) is another fully nonlinear Bayesian prediction/classification technique [32, 33, 34, 35, 36, 37, 38, 29, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51]. The GP treats all the data points as Gaussian (normal) distributions. Predictions are made through a covariance model of the data, representing the relationship between input data points and output data points. The GP is the equivalent of an infinite node, single hidden layer neural network [33], and is capable of representing the same types of behavior that neural networks are. Additionally, for problems with small numbers of data points, the training of a Gaussian process is significantly less time consuming than that for a neural network. The downside of Gaussian processes is that training and prediction time scales poorly with increasing data points.

2.1.5.2 Uses of Meta-modeling

In addition to replacing complex analysis tools in a design space exploration, meta-models can be used to extend the fidelity of a lower fidelity analysis, enable visualization environments, such as *JMP*'s contour profiler environment, and allow inverted prediction of parameters using the system responses. The ability to increase the fidelity of a lower fidelity analysis and to enable easy visualization of results are often

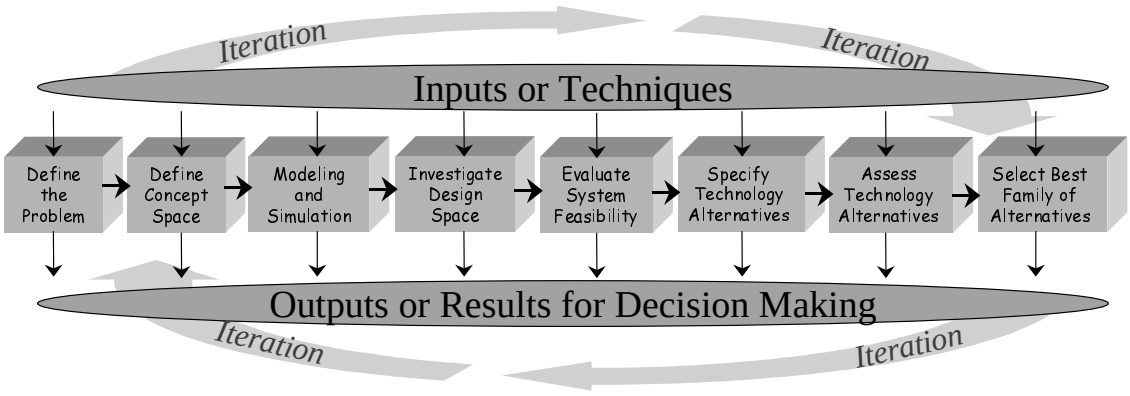


Figure 19: Basic TIES Methodology Flowchart [54]

of the most interest. In the case of RCD, the use of meta-models as a visualization enabling technology is essential.

2.1.6 Technology Identification Evaluation and Selection

Technology Identification, Evaluation, and Selection (TIES) was developed as a means of identifying which technologies were most appropriate to expand or create a feasible and viable design space [10, 52, 53, 54, 55]. The TIES methodology, which is illustrated in Figure 19, involves the investigation of the feasibility and viability of the initial design space using the techniques mentioned above. Additionally, if necessary it prescribes a method to investigate, evaluate, and determine which technologies can be "infused" to open up a design point.

The TIES methodology, as developed at Georgia Tech, attempts to incorporate the effect of uncertainty related to the development process of a technology on the performance and risk associated with that technology [10]. Critical to the development of the TIES methods is the identification of potential technologies, their influence on the system being studied, compatibility with other technologies, and costs and risks associated with the further development of beyond SOA technologies. This can be accomplished with methods such as Technology Impact Forecasting (TIF)

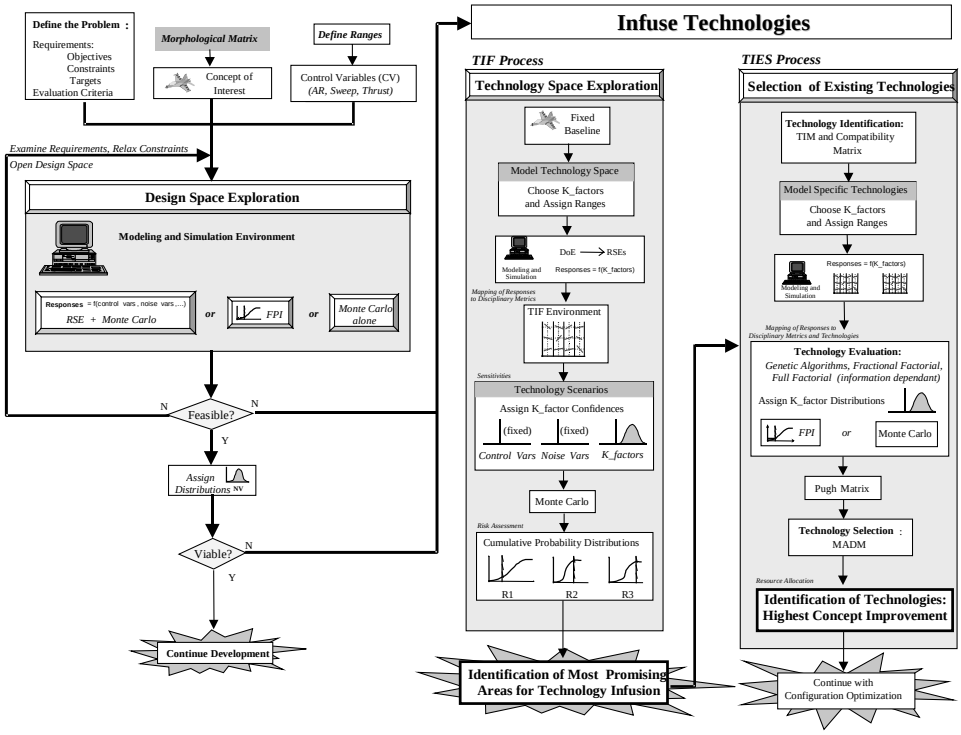


Figure 20: Design Space Exploration and System Feasibility and Viability Assessment Process [17, 59]

[20, 56, 57, 58, 59]. With both TIES and TIF included, the revised system feasibility & viability assessment environment is shown in Figure 20. The use of TIES has been extremely successful in furthering the ability to investigate technologies, and ultimately in implementing the type of environment envisioned for VSLCD. TIES has even been adapted to deal with large, complex, sets of technologies [60].

2.1.7 The Unified Tradeoff Environment

The necessity of investigating the effect that requirements have on the design of a system, one of the cornerstones in the development of RCD, led to the development of the UTE [1, 2, 3, 4, 5]. The purpose of the UTE is to combine the effects of mission requirements, vehicle attributes, and technologies into one environment [2]. A notional UTE implementation is shown in Figure 21.

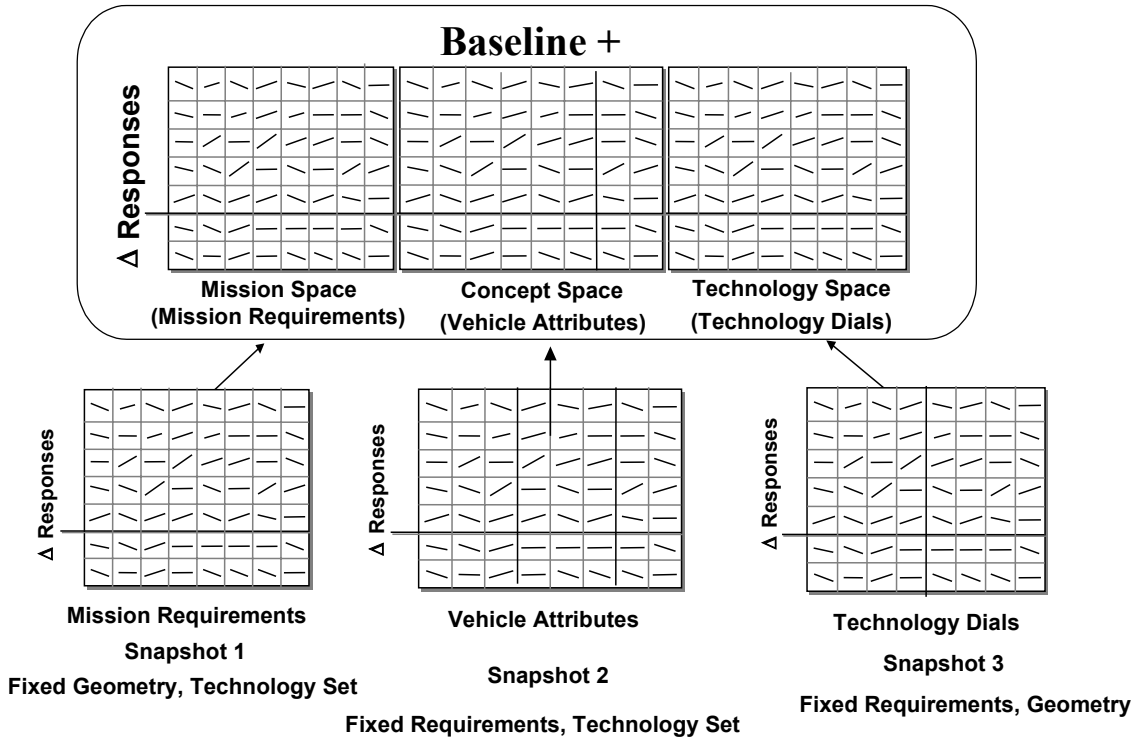


Figure 21: Notional Unified Tradeoff Environment [1, 2, 3, 4]

The UTE treats the design requirements the same as any other design variable/attribute, incorporating them as multipliers on the baseline vehicle. Critical to the implementation of UTE is that the change in requirements produces only an evolutionary change in the system under study. The UTE has been applied to both the US Army's Future Transport Rotorcraft & the development of the F/A18-E/F [2, 1, 3, 4, 5]. Even with the evolution of the requirements, the basic behavior of the system did not change. Since the UTE is focused primarily upon "evolving" systems with changing requirements it cannot be used for a broad investigation of the requirements (hyper)space.

2.1.8 Additional Advanced Design Methods

In addition to the methods detailed so far in this section, there exist several other methods that are substantially similar to those presented above. Additionally, design

engineers use many of the standard Multidisciplinary Design Optimization (MDO) techniques. Some example applications of MDO include:

- Optimization of a tiltrotor wing [61]
- Sizing of stopped rotor configurations [62]
- Aircraft preliminary design [63]
- Conceptual aerospace vehicle design [64]
- Space vehicle trajectory optimizations [65]

The field that MDO encompasses is quite broad and many sub-specialties have spawned. One of the more prevalent is Collaborative Optimization CO [66, 67]. CO has been applied to reusable launch vehicles [68], space based infrared systems [69], etc. Another advanced design method, that focuses on a computational problems solving environment is the Problem Solving Environment (PSE) developed at the University of Southampton [70]. The perturbations of these modern design methods are many and varied; however, they all rely on a similar understanding in the problem.

2.1.9 Comments on Applicability of Current Modern Methods

While the development of IPPD, RDS, TIES, etc. has enabled an improvement in design for affordability, and an understanding of technology development, they are not the end game of the development of modern design methods. The use of "frozen" methods such as the inherent use of a baseline vehicle in the traditional TIES methodology, plus the treatment of requirements as either constraints, or similar to design variables, limits the ability of the methods to incorporate truly radical new technologies. These methods are useful when a system has been settled upon and a higher fidelity approach is desired. Further the understanding of which type of system is most appropriate given a set of requirements, and what is the risk associated

with choosing a particular system with the knowledge that the future path of the requirements is itself uncertain. This has led the author to investigate a section of mathematics often used in the social sciences, economics, and the biosciences: Catastrophe Theory.

2.2 Bifurcation & Catastrophe Theories

Typically complex systems design has been approached as a **fully** deterministic system. However, a large amount of empirical evidence suggests that this is not the case. As described in Chapter 1, the requirements for the design of a system are not entirely static. Further, the actual values of these requirements may not be fully known at the time of program initiation. This nondeterministic behavior only serves to make the entire problem even more complex. Furthermore, most systems are nonlinear in nature. Nonlinear systems with no deterministic initial and boundary conditions, commonly exhibit instabilities and nonsmoothness with rapid changes in behavior for a change in one or more of the system parameters. This behavior is known as a bifurcation, specifically defined as:

Definition 2. A rapid change in the type of system dynamics when parameters in the system are varied is known as a **bifurcation** [71, p. 299].

Bifurcations are well known in aerospace engineering, one of the most common examples being the *Hopf Bifurcation*, which describes the transition from a stable system to a limit cycle. Aeroelastic divergence and flutter also occur at a bifurcation point. Each of these bifurcations are members of a specific class of bifurcations, those for which both the state and behavior change at the bifurcation point. Of course, the recognized prevalence of bifurcations in systems does not inherently lend any more insight to our understanding of the system behavior, especially since most of the "simplified" models used are designed to avoid the presence of bifurcations. However, the application of a subset of bifurcation and singularity theory, known

as **Catastrophe Theory**, shows significant promise in increasing our knowledge of complex systems design behavior with respect to the design requirements.

2.2.1 Catastrophe Theory

Catastrophe Theory, first promulgated by René Thom in *Stabilité Structurelle et Morphogenèse* [72], deals specifically with the classification of certain types of critical points of smooth functions. The behavior of these critical points result in the bifurcations that Thom calls "elementary catastrophes [73]." The mathematical formulation of Catastrophe Theory is presented in Appendix A. In the most rigorous form the "elementary" catastrophes are solved analytically. These catastrophes are derived from polynomial "potential" functions that consist of two types of variables, the **Control Variables (CV)** and the **State Variables (SV)**. They are defined as:

Definition 3. A variable, which when changed alters the behavior of a functional system is called a **control** variable [71].

Definition 4. A variable that **functionally** determines the state, i.e. the response, of a system is called a **state** variable [73].

One of the inherent problems with using analytical catastrophe theory is that the dimensionality of the problem is limited [74, p. 19]. The exact reason for this is discussed in Appendix A. However, all of the catastrophes possess a series of properties called *flags*, these flags allow for numeric solutions of problems with nonanalytical solutions and problems of higher dimensions. These flags are also discussed in more detail in Appendix A.

2.2.2 Applicability of Catastrophe Theory

If one accepts that complex system design is highly dimensional and the response is of a medium to high order, the question of whether or not Catastrophe Theory is truly usable becomes one of importance. Typically a complex system will possess several

control variables, and potentially thousands of state variables. Since the number of catastrophe forms becomes infinite for nonquadratic co-ranks, SVs, greater than two and co-dimensions, CVs, greater than five, it is possible to obtain analytical solutions only for a small subset of complex system design problems. However, there is no inherent requirement for an analytical solution. Just as aerodynamicists have accepted the lack of full analytical solutions to the Navier-Stokes equation, design engineers can accept the lack of analytical catastrophe theory solutions. Catastrophe theory has been commonly applied to economics, biosciences, and social sciences; fields where well defined analytical models are often not available. Sample applications include:

- Speculative market bubbles and crashes [74, pp. 26-27]
- Zeeman's stock market model using heterogeneous agents [74, pp. 26-27]
- Attachment behavior & arousal [75]
- Memory in enzyme membranes [76]
- Thyroid dysfunction [76]
- Gates of animals [77]
- Centralization of growth and decline of organizations [78]
- Decision framing [79]

Catastrophe theory models have also been used in several applications for both system dynamics [80, 81] and fluid dynamics [82, 83, 84]. Additionally, catastrophe theory has influenced probability theory, a key component of modern robust design methods [85]. Given the broad applications of catastrophe theory, both inside and outside of engineering, the question becomes, not whether Catastrophe theory can be applied, but rather how to apply it.

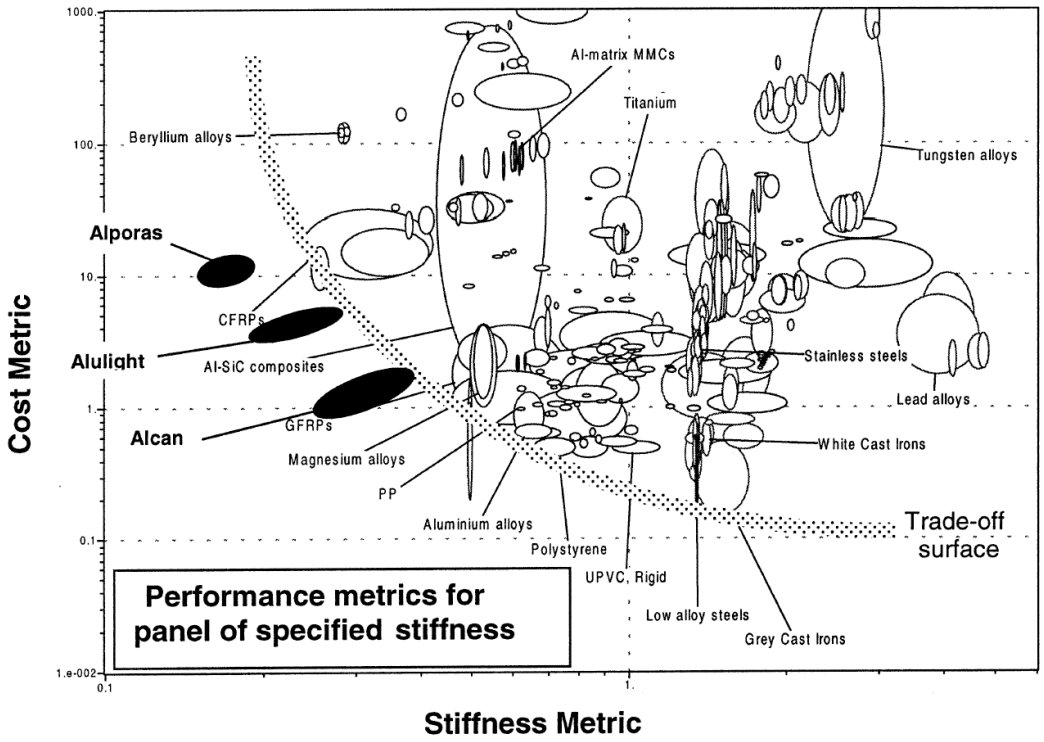


Figure 22: Ashby Chart for Performance Cost Trade-off [88]

2.3 Materials Science Visualization Techniques: The Ashby Chart

The application of catastrophe theory and the investigation of the system requirements space places a distinct burden upon finding a means to visualize the results. Because of the large number of material types available, all with different properties, material scientists have developed a graphical means of visualizing materials and material families on two-dimensional plots of particular metrics. These plots, known as Ashby charts, were developed by Michael F. Ashby of the University of Cambridge [86, 87, 88]. Figure 22 shows an example of an Ashby chart.

The primary benefit of the Ashby charts is that they provide a method of displaying different materials and material families in the plane of two properties. This can be done as a slice through single values of other properties, or as a projection of all families onto a two-dimensional surface. Incidentally, the Ashby charts look very

similar to the notional requirements space charts shown in Figures 2 and 4 through 8.

Another feature of the Ashby charts, as shown in Figure 22, is that a "trade-off surface", also known as a Pareto front, can be superimposed. This surface represents the "best" that a given level of materials technology can produce. This capability is a natural outcome of the manner in which an Ashby chart is constructed. The benefit here is that mapping to the requirements as an extension to system design, can quickly show the Pareto front for a given set of requirement classes. In the case of systems design, specific design types or families would be projected onto these two-dimensional planes of the requirements space.

2.4 Pareto Fronts

The Pareto Front is based on the Pareto principle from microeconomics. This principle describes a Pareto efficient condition, which is defined as:

Definition 5. Any allocation such that any reallocation would harm at least one person is said to be **Pareto efficient** [89, p. 330].

The Pareto front in engineering describes the "best" that can be achieved in any set of metrics. If one wants to improve the system's performance in a single metric it is necessary to degrade the system's performance in at least one other metric. The use and determination of Pareto fronts are common in multi-objective optimization where the exact weighting of the sub objectives in the main objective function may not be known [90, 91, 90, 92, 31, 93, 94].

In aerospace engineering, the use of the Pareto front, or it's functional equivalent is becoming more common. In conjunction with the TIES method, it is known as the "technology frontier [55]." The UTE can be used to determine the "mission space frontier" and the compromised designs [4]. This is notionally shown in Figure 23. The Pareto front can also be used to determine the technology risk benefit frontier

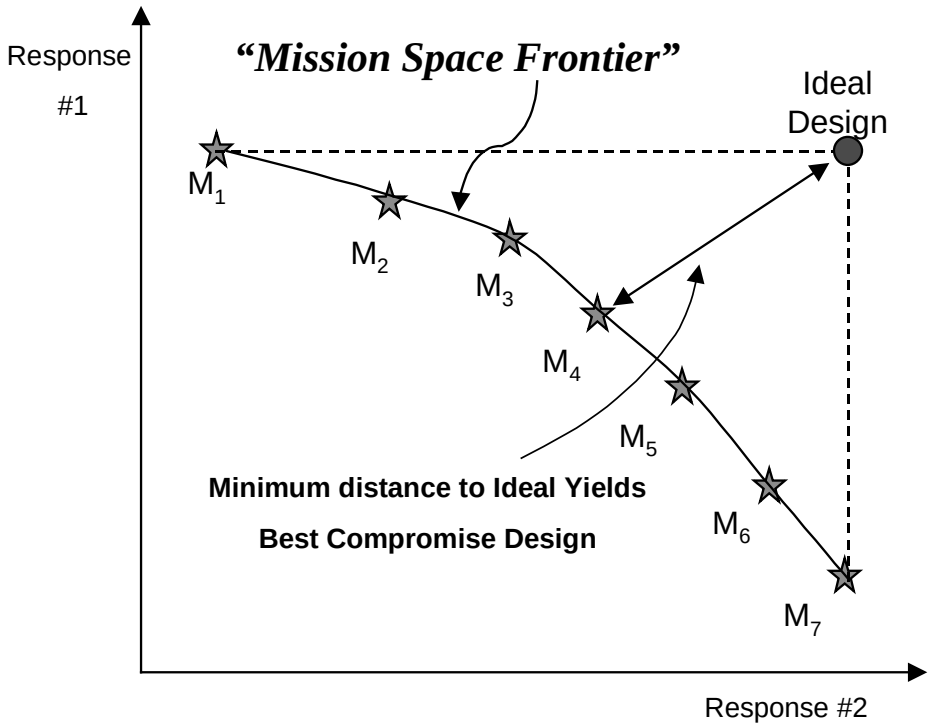


Figure 23: Notional Mission Space Pareto Frontier [5]

[95], this is shown in Figure 24. In the case of RCD, the Pareto frontier is the "best" set of settings that the requirements can have at a given technology level.

The combination of Pareto fronts with conceptual design is a rapidly advancing field. Mattson and Messac have proposed and exampled a Pareto front based concept selection method they call "s-Pareto Frontier" [96, 97]. The s-Pareto methodology, is essentially an extension of typical "frozen" design and MDO methods to include the ability to search for a Pareto frontier with or without uncertainty. It does not uncouple the state and control variables, which is needed to truly explore the design space. However, the success of the s-Pareto method does indicate that the goals for the RCD methodology are reachable. Particularly with the combination of search algorithms and visualization.

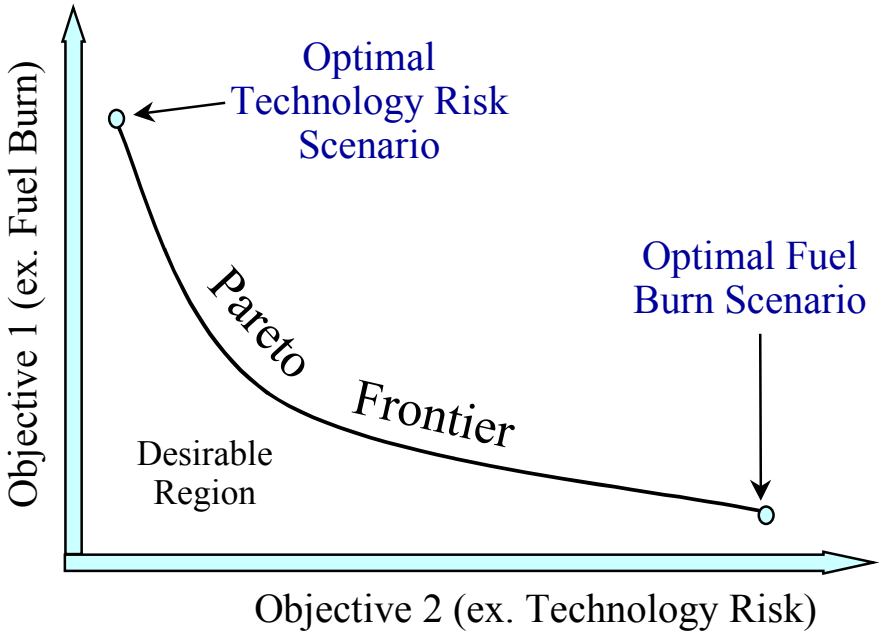


Figure 24: Notional Engine Technology Risk Pareto Frontier [95]

CHAPTER III

HYPOTHESES & RESEARCH QUESTIONS

3.1 Hypotheses

The use of a combination of traditional & modern design techniques, plus an understanding of both the qualitative and quantitative properties of catastrophe theory enable several hypotheses to be drawn with respect to modern complex systems design. These are listed below:

Hypothesis I: Complex systems are governed by two primary types of variables:

Control Variables: These control the behavior of the system. These are the system requirements.

State Variables: These define the actual response of the system. These are the system design variables.

Hypothesis II: Complex systems are composed of multiple levels of subsystems and these subsystems can themselves be viewed as systems (Hierarchical Decomposition).

The control and state variables for a higher level system become the control variables for a lower level subsystem.

Hypothesis III: Discontinuities exist in system states due to *technological limits* in component subsystems.

These discontinuities possess the same properties (flags) as the "elementary catastrophes".

Hypothesis IV: Determination of the discontinuity boundaries is both possible and computationally feasible.

Hypothesis V: A graphical mapping of multiple system types from the state (design) onto the control (requirements) space is possible in a method similar to that developed by Michael Ashby, in order to classify material types and families.

Hypothesis VI: It is feasible to determine a "Requirements Pareto Front" for any set of all possible systems.

3.2 Explanation of Hypotheses

The hypotheses listed above make up the basic foundation of what can be described as Requirements Controlled Design (RCD). This new method of design will free up design choices, enable "out of the box" thinking, and evaluate truly revolutionary technologies and concepts. However, the ideas contained within RCD may not be intuitive to most complex system designers. A more in-depth discussion of the hypotheses is, therefore, appropriate.

3.2.1 Hypothesis 1: Control & State Variables with Respect to System Design

Conventional wisdom holds that design variables are inherently the control variables for complex design. However, given a rudimentary understanding of the qualitative behavior of bifurcation and catastrophe theories it quickly becomes obvious that the design variables of a system and all of its subsystems and components, actually **functionally** describe the unique state of the system. This represents the functional **germ** of a system design problem. Therefore, using Definition 4, on page 32, which states that state variables define the functional response of the system, the design variables must be the **state** variables. This begs the question, what are the control

variables? This can be answered, by using the functional relationship given in Equation 47 on page 173, i.e., CVs define the general behavior not the complete state of the system. Since the design variables are the SVs, then it stands to reason that the CVs are those variables upon which the design variables are dependent. Again, conventional wisdom holds that the designer "independently" sets the design variables. Truthfully, the design variables are determined by the requirement to minimize or maximize some combination of figures of merit, F. Ideally, this combination of figures of merit produces a smooth function; however, there is no requirement that this is the case. This would make the final setting of the design variables, p, a critical point of F. Therefore, one needs to look for families of potential perturbations of the function F. More specifically, a need exists to look for the families of perturbations that meet the definition of **versal**, as given in Definition 19 on page 174, i.e., they describe all of the possible behaviors of the system. The question then becomes what other variables exist that can perturb the system response, and upon which variables would the design variables themselves be dependent?

One possible answer to this question is the system requirements. It is known that different settings of system requirements produce significantly different systems. Additionally, there are, in many cases, multiple final solutions to a single setting of the requirements. Using these two properties and comparing them to the definition of CV given in Definition 3 on page 32, it can be quickly determined that the requirements are, in fact, control variables. Furthermore, according to Arnol'd, it is typical for a system to be highly dimensional in SVs, $\mathbf{R}^n$, and of much lower dimension in the CVs, $\mathbf{R}^l$ [98]. This tends to track with the respective dimensionalities of design variables and requirements in complex system design, i.e. there may be thousands of design variables, but only tens of requirements.

3.2.2 Hypothesis 2: Hierarchical Decomposition of Complex Systems

As stated above, the design variables for a complex system can easily number in the thousands when all of the subsystems and components are taken into account. Obviously, it quickly becomes impossible to investigate even a small portion of all of the possible settings of the design variables if this is the case. There is, however, a way to simplify the task involved significantly. It is common for modern organizations to subdivide the development task of a complex problem, be it computer software or aircraft hardware, into different areas of responsibility and even different contractors. The most obvious examples of this, with respect to aerospace engineering, is the sub contracting of avionics and power-plant subsystems to different contractors, i.e. Honeywell, Rockwell, General Electric, Pratt & Whitney, or Rolls Royce. In many cases this division is so complete that only a small interface needs to be defined for the higher system to make use of a subsystem. For example all of the major commercial and noncommercial computer operating systems provide a means for application developers to access core functionality. This is called an Application Programing Interface (API). Similarly, when GE or Pratt deliver a jet engine to Boeing or Airbus, there is a specific set of structural, mechanical, electrical, and fluidic interfaces that the airframer and engine manufacturer have agreed upon. It, therefore, stands to reason that a complex system, during the design process, could also be subdivided in a similar manner. A flow-chart for this method is shown in Figure 25.

This is inherently a good idea; however, modern design engineers typically think of all of these systems as interconnected and iterative. While this is true, a quick examination of techniques used in modern design methods, shows that a form of the decomposition method is currently in use: The manipulation of "k" factors in the TIES methodology. The "k" factors are inherently modifiers of the subsystem level metrics or properties. Therefore, it is not required that a complex system be broken down into all of its components. Instead each of the major subsystems can be

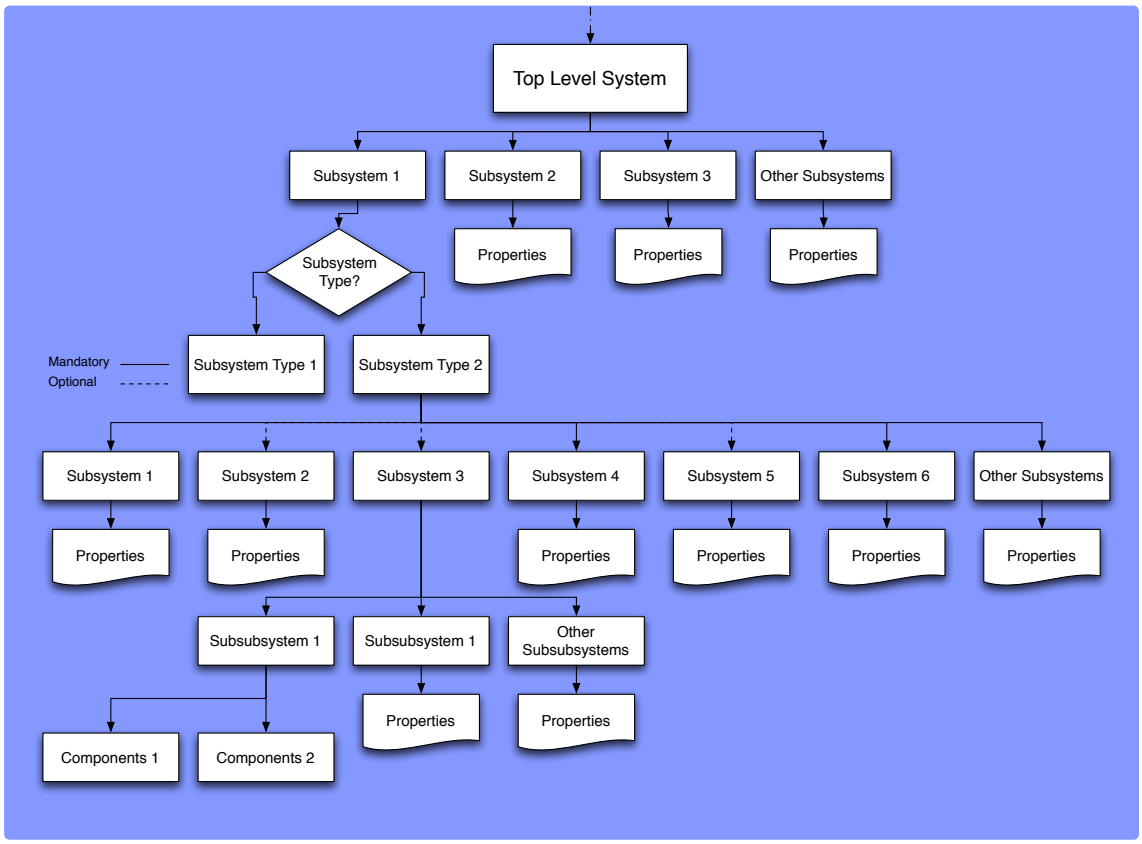


Figure 25: Aerospace System Hierarchical Buildup

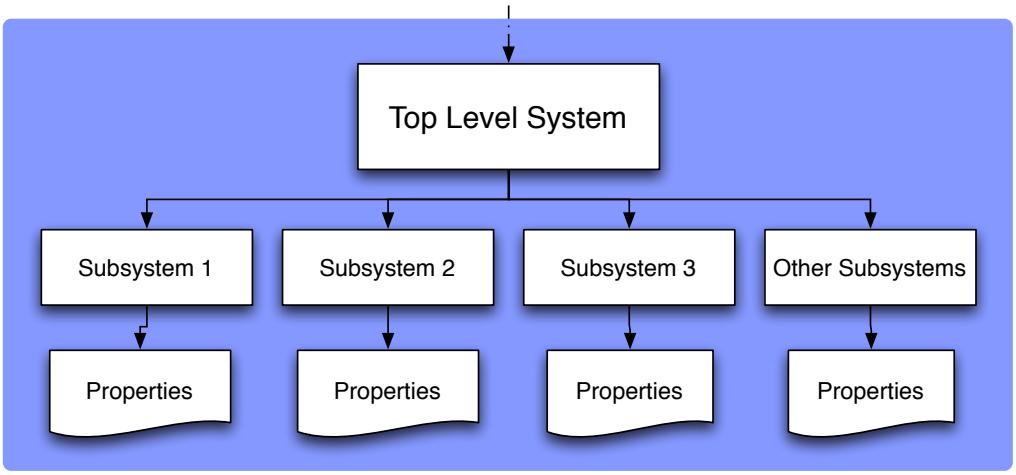


Figure 26: Simplified Aerospace System Hierarchical Buildup

represented by its metrics or properties. This simplified hierarchical buildup is shown in Figure 26. Further, the validity of using this approach to investigate the effect of requirements on a vehicle system has been demonstrated by the author [6, 7].

3.2.3 Hypothesis 3: Existence and Properties of System State Discontinuities

Since most modern design is performed by examining small perturbations of the design or state variables about a baseline point, it is easy to come to the conclusion that the system response is fully continuous. However, a working understanding of both the physical principles and limitations which apply to any system, and the current or future state of the art of technologies that go into it, indicates that if the perturbations become large enough, the baseline assumptions no-longer hold. Looking at historical design decisions also supports this claim. There were no gas-turbine powered, jet-propelled aircraft prior to the second world war because the technology state of the art did not allow for this type of aircraft. During the war, however, the need for significantly faster fighter aircraft drove the development of significantly improved jet engines, enabling the Me-262, Gloster Meteor, etc. Empirically the existence of technology boundaries does not, however, provide an understanding of

those boundaries.

Envision an environment where any one of the system requirements can be varied, over any range, at will, with the response updating in real time. In this ideal environment, all of the technology boundaries, be they imposed by current SOA or by some physical limit, e.g. the second law of thermodynamics or the speed of light, would be instantly incorporated, the system would always seek the "best" state solution, and the "system type" state variable would change accordingly. Of course, this environment does not exist today; however, using a reduced complexity system decomposition similar to the one shown in Figure 26, it should be possible to determine and understand these boundaries in the requirements space.

3.2.3.1 Properties of System State Discontinuities

The applicability of catastrophe theory in complex systems design depends upon the properties of the System State Discontinuities. The list of the properties of the "elementary" catastrophes given on page 177 contains the "flags" that determine the validity of describing a system state discontinuity as a catastrophe, these flags are also listed below.

Bimodality: *Can the system exist in multiple distinct equilibrium states? Can there be two or more solutions for a given set of requirements, e.g. can an aircraft and missile both fulfill the requirements.*

Inaccessibility: Are there points in the space that cannot be reached? Can the "backside" of a system state be reached when there is overlap between two or more system states

Sudden Jumps: *Do small changes in the requirements produce discontinuous changes in the system state?*

Hysteresis: Does the transition point between system states occur at a different

point, dependent upon from which direction it is approached?

Divergence: *Do small changes in one requirement produce vastly different system states when another requirement is then varied?*

The presence of these properties, in general, is relatively straightforward to demonstrate. At a minimum, two methods present themselves. The first is to create an analytical "potential" equation, i.e. some sort of overall evaluation criteria, or multi-attribute decision making equation, using the design variables. This equation would then be perturbed using the functional effect of the system requirements. This, means that some sort of algebraic, differentiable representation of the effect of the requirements on the "potential" function must be known. The other option is to perform a numerical simulation of the requirements space, optimizing the "potential" function to determine the space topography. Both of these techniques have their advantages and disadvantages which will be discussed later.

3.2.4 Hypothesis 4: Computational Feasibility of Determination of System State Boundaries

One of the unfortunate properties of analytical "elementary" catastrophe theory is the limited maximum size of the system codimension and corank. This limitation, first mentioned in Chapter 2, is that the number of catastrophes tends toward infinity rapidly when the codimension is greater than four, and the corank is greater than two. This is not a problem if the system response to all but two state/design variables is linear or quadratic; coupled with a small number of controlling requirements. As stated earlier, this situation occurs relatively rarely, generally only in trivial/academic exercises. The use of a numerical method, identifying system state transitions that fit the catastrophe flags is of far greater real-world practicality. However, in the most general form, a significant survey of the design space needs to be undertaken. With a reasonable number of requirements and design variables, this can become extremely

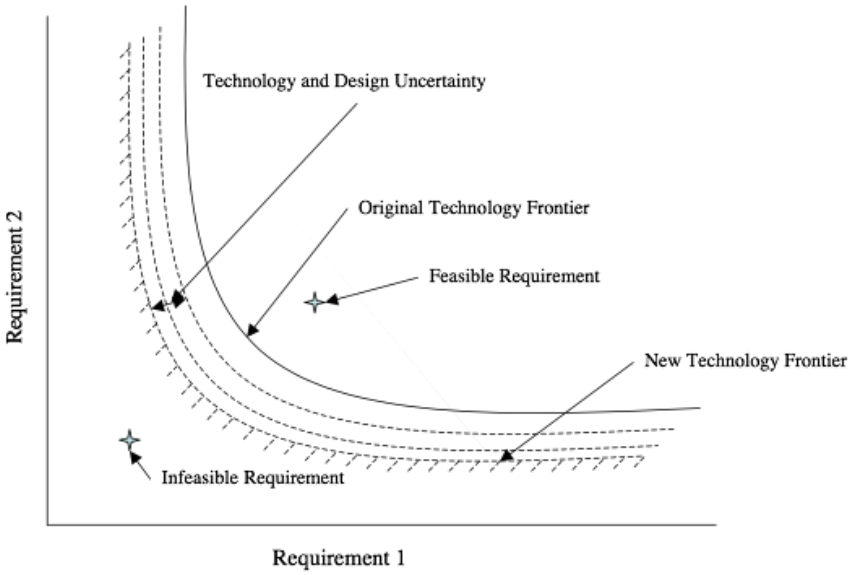


Figure 27: Notional Representation of the Requirements Hyperspace

computationally intensive. A significant portion of this research will be spent on possible remedies to this problem.

3.2.5 Hypothesis 5: Graphical Representation of the Requirements Space

The Ashby chart shown in Figure 22 on page 34 shows material families on a two-dimensional metric plane. Using either method described previously it is possible to get a representation of the system states as they exist in the requirements space; either analytically or numerically. With the data it is simply a matter of plotting the data/equations to produce the desired graphical result. The actual acquisition of the data is of greater significance and difficulty. A notional representation of the requirements (hyper)space is shown in Figure 27.

3.2.6 Hypothesis 6: Determination of Requirements Pareto Front

The process of determining single system state boundaries, naturally leads to the desire to investigate the overall state boundaries for multiple systems. By combining techniques required to to solve the computational complexity issue addressed above,

with a matrix of possible components and their properties, it should be reasonably straightforward to investigate the overall "requirements Pareto front."

The requirements Pareto front is defined as the set of specific requirements where an increase in the severity of one requirement requires the relaxation of one or more other requirements. This is identical to what Ashby calls the trade-off surface shown in Figure 22 [88].

3.3 Research Questions

In order to properly prove the above Hypotheses the following questions need to be asked.

1. Can an analytical "potential" equation be developed for complex aerospace systems design, and if so, can the co-dimension and the nonquadratic co-rank be limited such that a purely analytical solution can be determined?

2. Is there a computationally efficient method to determine the system state boundaries in the requirements (hyper)space?

    - Can an Evolutionary Algorithm (EA) provide this capability?

    - Is a Pareto seeking EA necessary and sufficient for this task?

3. Does an Ashby style chart provide a sufficient visualization technique to allow quick identification of the system state boundaries in the requirements (hyper)space?

    - Is it possible to create an easy to comprehend, dynamic visualization environment using the Ashby style chart and meta-models?

CHAPTER IV

SOLUTION APPROACHES

In the search for a solution each of the hypotheses and research questions given in the previous chapter was addressed. Hypotheses 1 through 3 were relatively straightforward to observe. These could be verified using the methods used for the evaluation of Hypothesis 4 and 6 and Research Questions 1 and 2. Further, work was performed in order to validate Hypothesis 5 and Question 3. The general method for RCD, as appropriate to answering Questions 1 through 3, is given in Figure 28.

4.1 Feasibility of an Analytical Solution (Question 1)

Using the knowledge gained in Chapter 2 and Appendix A, the problem inherent with determining if and how to analytically solve for the catastrophe locations in a complex system design is threefold. First, one must create a sufficiently smooth "potential" function, containing the necessary state variables, the optimization of which leads to

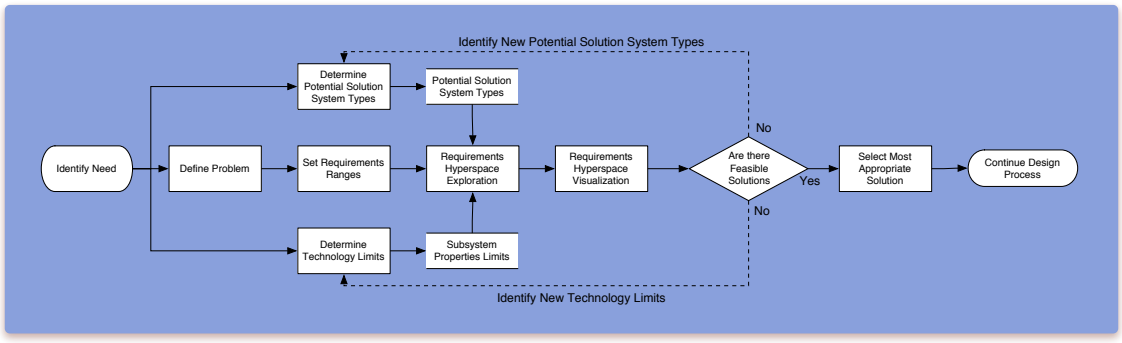


Figure 28: General Flow Chart for Implementation of Requirements Controlled Design

the "best" solution. Second, the necessary versal unfolding to this equation needs to be determined, incorporating system requirements as control variables. Third, the necessary requirements cannot realistically number more than five, and the number of non-quadratic design variable effects cannot number more than three. Identification of a potential test case is itself inherently difficult, even before the potential perturbations are identified. This is a particularly large problem in most complex systems, where the numbers of design variables, technology metrics, and requirements is in the thousands to millions. Therefore, any attempt at an analytical approach must use a significantly simplified model. These simplified models introduce, in addition to the problem of lower fidelity, their own set of problems. As an example a simple aircraft sizing problem was analyzed.

4.1.1 Simple Aircraft Sizing Example

One of the primary steps in sizing a new aircraft is to virtually "fly" the mission. In simple conceptual design the mission is broken down into segments and each of these segments is analyzed and the mission reassembled. The primary goal of this process is to determine the ratio between the end and beginning of the mission segment vehicle weight. Since each mission segment is a relatively self-contained unit it is practical to analyze a single segment for the existence of catastrophes.

To undertake this investigation it is useful to look at all of the assumptions that are typically included in a basic aircraft sizing example. Using the basic point mass equations for aircraft flight, given in Equations 2 and 3 [99, pp. 34], as a starting point.

$$T - D - W\sin(\gamma) = 0 \tag{2}$$

$$L - W\cos(\gamma) = 0 \tag{3}$$

Where T is thrust, D is drag, W is weight, L is lift, and γ is the flight path angle with respect to the inertial frame. These simplified equations work well for simple

cruising flight were $\gamma \approx 0$. These equilibria are good at describing why an aircraft stays in the air, but they say little about where it is going. To answer this we add Equations 4 and 5, to the above equations.

$$\frac{dX}{dt} = V_g \cos \gamma_g \tag{4}$$

$$\frac{dh}{dt} = V_g \sin \gamma_g \tag{5}$$

Where X is distance along the ground, h is height above the ground, V_g is ground speed, and γ_g is the path angle with respect to the horizon. Ground speed is the combination of the velocity of the aircraft and the velocity of the air mass.

There are several assumptions already included in these equations, not the least of which is that the thrust line is aligned with the inertial frame. Adding more assumptions including that the aircraft is "cruising", i.e., the flight path angle is very small, $\gamma \to 0$, $\cos \gamma \to 1$, and $\sin \gamma \to \gamma \to 0$, Equations 2 and 3 simplify significantly.

$$T = D \tag{6}$$

$$L = W \tag{7}$$

Additionally, making the common assumptions of still air, $V_g = V$, and a flat earth, $\gamma_g = \gamma$ Equations 4 and 5 simplify greatly.

$$\frac{dX}{dt} = V \tag{8}$$

$$\frac{dh}{dt} = 0 \tag{9}$$

If it is assumed that the airplane burns fuel to produce thrust we add Equation 10,

$$\frac{-dW}{dt} = Tc \tag{10}$$

where c is known as the specific fuel consumption of the powerplant, a subsystem metric. Additionally it is known that T, D, and c are themselves functions of other variables.

$$T = f\left(h, V, \mathbf{X}\right) \tag{11}$$

$$D = f\left(h, V, W, \mathbf{X}\right) \tag{12}$$

$$c = f\left(h, V, \mathbf{X}\right) \tag{13}$$

Where $\mathbf{X}$ is the vector of vehicle state/design variables. Assuming that $\mathbf{X}$ is held constant, D and T become functions of h, V, W only. Remember that W is the instantaneous weight, which is itself dependent, but not entirely determined by $\mathbf{X}$.

As another simplifying assumption, we add the quadratic drag polar:

$$D = C_D q S = \frac{1}{2} C_D \rho V^2 S \tag{14a}$$

assuming subsonic flight

$$C_D = C_{D_0} + C_{D_i} \tag{14b}$$

$$\tag{14c}$$

$$C_{D_i} = K_1 C_L + K_2 C_L^2 \tag{14d}$$

and

$$C_D = C_{D_0} + K_1 C_L + K_2 C_L^2 \tag{14e}$$

Therefore, D can be represented by Equation 15,

$$D = \rho V^2 S \left(C_{D_0} + K_1 C_L + K_2 C_L^2\right) \tag{15}$$

and C_L by,

$$C_L = \frac{L}{qS} = \frac{L}{\frac{1}{2}\rho V^2 S} = \frac{2W}{\rho V^2 S} \tag{16}$$

Drag is still a function of V, h, and W but is now more defined. At this point it is useful to define another variable, i.e. the instantaneous lift to drag ratio $\frac{L}{D}$, which is given in Equation 17, assuming $K_1 = 0$.

$$\frac{L}{D} = \frac{1}{\frac{C_{D_0}\rho V_2}{2}\frac{S}{W} + \frac{2K_2}{\rho V_2}\frac{W}{S}} \tag{17}$$

Note that K_2, C_{D_0}, and S are all functions of $\mathbf{X}$.

Now if we want to determine the fuel burn for a differential range, taking Equations 6 through 10 and manipulating them we get Equation 18:

$$\frac{-dW}{dt} = Tc = Dc = \frac{D}{L}Lc = \frac{D}{L}Wc \tag{18a}$$

$$\frac{dX}{dt} = V \tag{18b}$$

and

$$\frac{dX}{-dW} = \frac{dX}{dt}\frac{1}{\frac{-dW}{dt}} = V\frac{L}{D}\frac{1}{Wc} \tag{18c}$$

These are the differential values for each. Typically the engineer want to determine the range for a given fuel burn, for which we use Equation 19.

$$X = \int_1^2 -\frac{V}{c}\frac{L}{D}\frac{1}{W}dW \tag{19}$$

We still want to see what elementary catastrophes exist; however, to make the problem workable we need to get a potential function. In this case the potential function H could be $H = \frac{W_{\text{fuel}}}{W_1}$. To get to this potential we need to make a few more assumptions:

1. $\frac{L}{D} = f\left(V, h, \mathbf{X}\right) = \text{constant}$
2. $V = \text{constant}$
3. $c = f\left(V, h, \mathbf{X}\right) = \text{constant}$

These are not necessarily accurate assumptions, but we are trying to make this problem manageable. Additionally, this is one of the "typical" cruise styles, called a "cruise-climb." Used properly the cruise-climb either maximizes the range or of a jet aircraft or it minimizes the time needed to fly a given distance. Therefore, Equation 19 becomes Equation 20, commonly known as the Breguet range equation.

$$X = \frac{V}{c}\frac{L}{D}\int_1^2 -\frac{1}{W}dW \tag{20a}$$

$$X = \frac{V}{c}\frac{L}{D}\left(\ln W_1 - \ln W_2\right) = \frac{V}{c}\frac{L}{D}\ln\left(\frac{W_1}{W_2}\right) \tag{20b}$$

Manipulating Equation 20b to get the segment weight fraction on the left hand side gives us Equation 21.

$$\frac{W_2}{W_1} = e^{-\frac{cX}{V\left(\frac{L}{D}\right)}} \tag{21}$$

Taking into account that only fuel is burned over the course of the mission segment, i.e. no payload is dropped, $W_2 = W_1 - W_{\mathsf{fuel}}$; therefore, $\frac{W_2}{W_1} = \frac{W_1 - W_{\mathsf{fuel}}}{W_1}$ and Equation 21 becomes,

$$H = \frac{W_{\mathsf{fuel}}}{W_1} = 1 - e^{-\frac{cX}{V\left(\frac{L}{D}\right)}} \tag{22}$$

Which is the potential function we will investigate for the occurrence of catastrophes .

To determine if any catastrophes exist, we need to take derivatives of H with respect to the state variables, c and $\frac{L}{D}$, set them to zero, and determine their dependence on the control variables, V and X. This is shown in Equation 23.

$$\begin{aligned} \frac{\partial H}{\partial c} &= \frac{X}{V\frac{L}{D}}e^{-\frac{cX}{V\left(\frac{L}{D}\right)}} \\ 0 &= \frac{X}{V\frac{L}{D}}e^{-\frac{cX}{V\left(\frac{L}{D}\right)}} \end{aligned} \tag{23a}$$

Since the equation is exponential there is no critical point, $\frac{\partial H}{\partial c} \to 0$ as $c \to \infty$.

$$\begin{aligned} \frac{\partial H}{\partial\left(\frac{L}{D}\right)} &= -\frac{cX}{V\left(\frac{L}{D}\right)^2}e^{-\frac{cX}{V\frac{L}{D}}} \\ 0 &= -\frac{cX}{V\left(\frac{L}{D}\right)^2}e^{-\frac{cX}{V\left(\frac{L}{D}\right)}} \end{aligned} \tag{23b}$$

Again, since the equation is an exponential there is no critical point, and the result is minimized as $\frac{L}{D} \to \infty$.

This poses a problem, we wanted to identify the analytical, elementary catastrophes present in a simple aerospace example; however, since the value of either c or $\frac{L}{D}$ that minimizes H is independent of all of the control variables there can be no

catastrophes. This is, strictly correct; however, it is all of the assumptions made between Equation 2 and Equation 22 which made this example trivial. In reality D and L are functions of V and h and other variables. The same holds true for the specific fuel consumption. It was only by assuming constant $\frac{L}{D}$, a quadratic drag polar and the like, that the relationships broke down. If one removes all of the assumptions made above the number of CVs and SVs present produces an infinite number of catastrophes, see Appendix A.

4.1.2 General Comments on the Feasibility of an Analytical Solution

The inability of the highly simplified sizing equations to produce analytical catastrophes is a direct result of the elimination of complexity from the problem. The reduction in the complexity eliminates the inherent dependencies of variables upon each other, and therefore, prevents them from influencing one another's settings in the minimization of the vehicle potential function. Another problem is the creation of an analytical potential function for a broad class or classes of vehicles. In the previous example the potential function was a highly simplified one. In many cases the potential function for an aerospace vehicle may not be so "neat" and may, in fact, be hidden by the underlying tools. Therefore, in order to allow for both of these problems, the lack of a simple potential function and the over simplification of a problem, it is possible to externally impose limits on the different state variables when using a numerical solution.

4.2 Computationally Feasible Numerical Boundary Discovery Method (Question 2)

Given the general lack of an analytical solution for the majority of problems of interest a numerical solution needed to be found. The typical use of computational codes in the conceptual and preliminary design of aerospace systems greatly simplifies the task facing the designer when trying to determine the locations of the catastrophic state

boundaries in a numerical manner. It should be possible to devise a general means to identify the system state boundaries. Several potential methods come to mind, including:

1. Grid Search
2. Local Optimization
3. Global Optimization

No matter what method is used, the theoretical or current SOA technology limits of the components need to be known. Additionally, either the numeric code needs to ensure that no "laws" of physics are violated or an external enforcement needs to be used. In the case of the grid search this can be applied after the fact; however, for both of the optimization methods, either constraints, or some form of penalty function need to be applied. In any case, a numerical solution requires that the technology limits mentioned in Hypothesis 3 be known for all of the components and systems being studied.

4.2.1 Grid Search Method

The first method, the grid search, is a "brute force" way of determining the boundary locations. In this case, a grid of points, of desired fineness, is distributed throughout the requirements space. Each point is evaluated in a fixed point iteration to achieve a closed solution. The requirements are matched and the subsystem properties are stored for later use. Since the grid is evaluated, no true search algorithm needs to be employed; however, the combinatorial size of the problem scales as l^n for a uniform grid, where n is the number of requirements, and l is the grid density. The grid search method is demonstrated in Appendix B.

4.2.1.1 Simple Example Using the Grid Search Method

Another grid-search example helps to describe the boundary location in a simple requirements (hyper)space. One of the common aircraft requirements is known as specific excess power (P_s). This requirement, which is usually specified for a specific Mach number, altitude, and point in the mission is a measure of the instantaneous rate of climb. Specific excess power translates into such metrics as service ceiling and fighter aircraft performance. It is typical calculated using Equation 24.

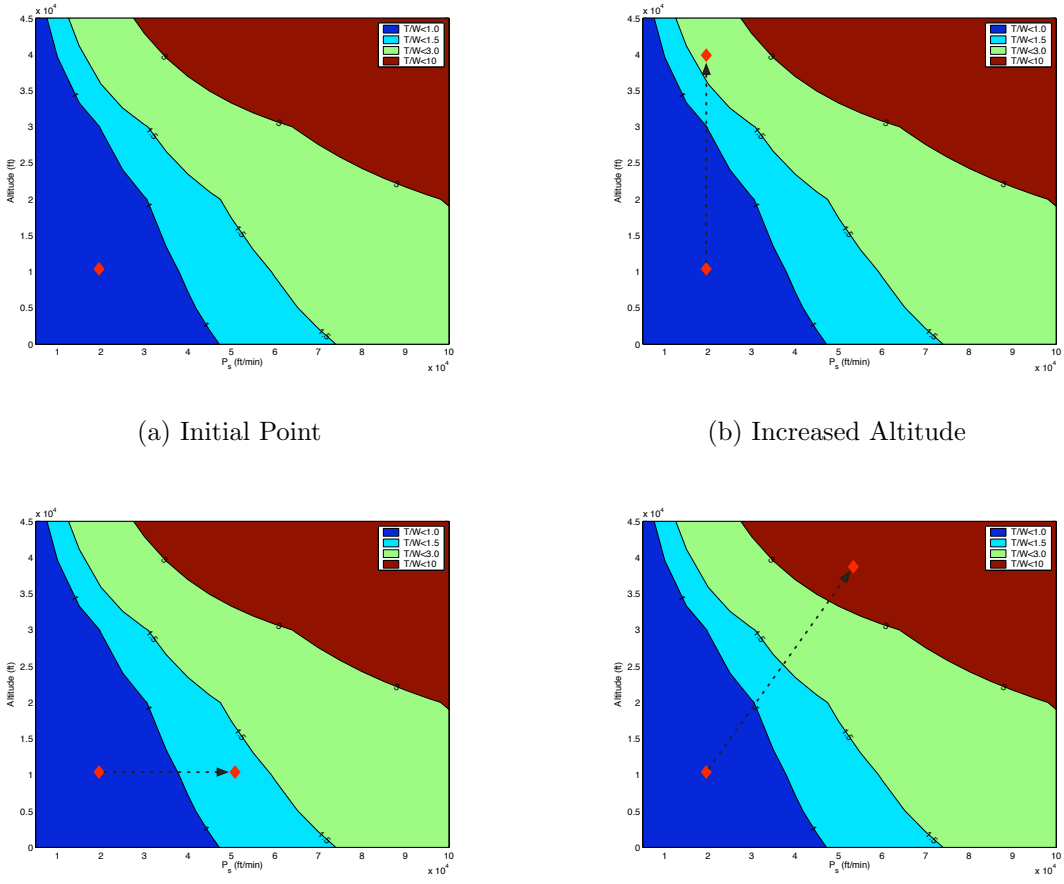
$$P_s = V\left(\frac{T}{W} - \frac{D}{W}\right) \tag{24}$$

This is a modification of Equation 2. Adding in the drag polar assumptions shown in Equations 14 through 16 and solving for $\frac{T}{W}$ it is possible to obtain Equation 25.

$$\frac{T}{W} = \frac{P_s}{V} + \frac{1}{2}\frac{\rho V^2}{W/S}\left[C_{D_0} + K\left(\frac{2}{\rho V^2}\frac{W}{S}\right)^2\right] \tag{25}$$

The nice feature of this highly simplified example is that the input, $\frac{T}{W}$, has been converted to the response. This places the requirements, P_s and altitude, in the equations parameters.

Assuming a constant $\frac{W}{S}$, Mach number, and aircraft state vector, but allowing $\frac{T}{W}$ to be the technology limit; it is possible to examine the P_s vs. altitude requirements space. This is shown in Figure 29. Assuming that there are four possible vehicles differing only in maximum $\frac{T}{W}$, 1.0, 1.5, 3, and 10, it is relatively easy to see the boundary contours for any combination of P_s and altitude. Obviously the $\frac{T}{W}$= 1.0 vehicle can meet a P_s= 20,000 feet per minute at 10,000 feet, shown in Figure 29(a), but not at 40,000 feet, shown in Figure 29(b). Conversely the vehicles with maximum $\frac{T}{W}$ limits greater than 1.5 can easily fulfill the P_s= 20,000 feet per minute at 40,000 feet requirement, also shown in Figure 29(b). If, instead, the P_s requirement is increased but the altitude held constant, shown in Figure 29(c), the vehicle is only required to have a $\frac{T}{W}$ greater then one. By combining both the increase in altitude



(a) Initial Point

(b) Increased Altitude

(c) Increased P_s

(d) Combination

Figure 29: System Type Model Predictions from COAX Model Cross Section Radius

and P_s, shown in Figure 29(d), the required $\frac{T}{W}$ jumps significantly, it must now be greater than three, leaving only the vehicle with a $\frac{T}{W}$ of ten as feasible. This simple example is similar to the notional example shown in Figures 2 and 3. The weakness in this example is that is does not allow the aircraft to equilibrate at each point. Therefore, the slice shown in Figure 29 contains no depth in the state/design space. This limits the usefulness of the example slightly.

4.2.1.2 Downsides to the use of the Grid Search Method

The problem inherent with the grid search method is that it requires a significant number of runs. In the example above there were four requirements, each investigated at six levels. This produced a minimum of 6^4 or 1296 runs per system. A quick look at Figures 79 through 84, on pages 183 and 187; however, indicate that 6 levels was not enough. If one uses a seven level system for the study the required number of runs increases to 2401, $\sim$ 1.9 times as many runs. More realistically the number of levels in each dimension would have to be doubled to get the desired resolution which would result in sixteen times more runs. Combine this with a more realistic number of requirements (ten or more) to be investigated and the number of required runs climbs even more substantially. In the case of a ten level, ten requirement hyperspace exploration the number of runs is ten billion. If each run takes half a second, the time to complete a two system study would be over 300 years. This is totally impractical. Additionally the storage requirements to keep track of all of the subsystem level properties would also grow exponentially. Obviously the grid search method can only be used for very simple problems, with few requirements. This is typically not the case in aerospace systems design. Therefore, another approach is necessary to discover the catastrophic requirements boundaries.

4.2.2 Optimization Methods

In contrast to the grid search, the other two options listed are both optimization methods. Figure 30, gives a graphical display of the general types of computational optimization schemes. The two optimization methods described herein fall under both the direct solution and penalty formulation families.

4.2.2.1 Local Optimization Methods

The tools for implementing a uni-modal optimization scheme are readily available and well known. They include the following:

- Powell's Method [101]
- Fletcher-Reeves Conjugate Gradient Method [102]
- Nelder-Meed Simplex Method [103]
- Sequential Linear Programing [104]
- Method of Feasible Directions [105]
- Sequential Quadratic Programing [104]

While most uni-modal optimization techniques are relatively straightforward and "inexpensive" to use, several constraints imposed by such tools limit their applicability. Since most uni-modal tools, especially those that are computationally efficient, use some form of line search, both the optimization function and penalty functions, if used, must be sufficiently smooth. If the limits are handled by constraints, these have to be properly defined, and for some of the techniques, must be sufficiently close to linear in behavior. If smoothness and continuity cannot be guaranteed, a less restrictive method needs to be employed.

The less restrictive methods, such as the Nelder-Mead simplex method, require similar levels of computational effort as the multi-modal optimization schemes listed

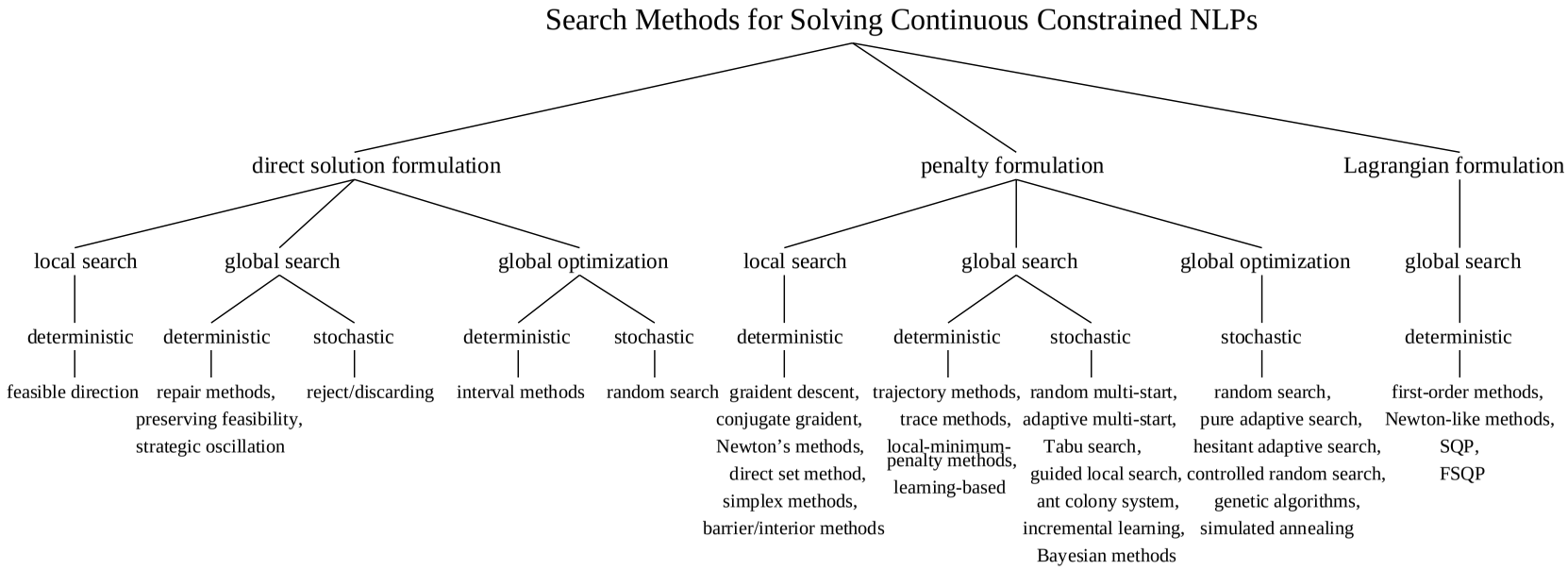


Figure 30: Optimization Families [100]

below. The Nelder-Mead method repeatedly performs a comparison of $(n+1)$ vertices, where n is the number of variables, of a general simplex, doing so until a minimum is reached [103]. Because the Nelder-Mead method assumes no specific behavior of the function being optimized, neither smooth nor continuous, it has the potential to be more generally applicable. However, the effectiveness of the Nelder-Mead routine can only be guaranteed for uni-modal functions. The author has demonstrated that uni-modality cannot be guaranteed for a complex aerospace design problem, and therefore, use of uni-modal methods is ill-advised [6, 7].

4.2.2.2 Global Optimization Methods

The class of methods that show the most promise of success, computationally and combinatorially, are the multi-modal optimization techniques. These include EA, Simulated Annealing (SA) [100, 106], and others. While each multi-modal technique has its advantages and disadvantages, they all provide sufficient capability to discover technology-driven state discontinuities. The use of an EA as the computational method of choice was arrived at, primarily, out of familiarity. The author has demonstrated the capability of a simple Genetic Algorithm (GA) to discover the requirements boundaries for a high-speed cruise propulsion system [107], as presented in Appendix C.

The use of a rudimentary GA for more complex problems, however, is less than satisfactory. This derives from the fact that the problem is one that not only requires a global approach, but also a multi-objective approach. Two papers by Coello and one by Zitzler, Deb, and Thiele review several of the more popular multi-objective EA approaches [108, 109, 110]. Two methods showed the greatest promise of combining multi-objective optimization and minimal computational expenditure. They were the Strength Pareto Evolutionary Algorithm (SPEA) [91] and a revision of the Nondominating Sorting Genetic Algorithm (NSGA-II) [111]. Another major benefit

of both of these algorithms was the availability of program skeletons and example source code.

Both of the Pareto EA/GAs were included in order to provide an alternate means in case one proved unsatisfactory to the task of discovering the technology limit boundaries. The SPEA was the first choice and was found to be an effective method for general problems. As discussed in the next chapter, SPEA proved satisfactory for the task.

4.3 Boundary Visualization Method (Question 3)

Translation of the data acquired from either a grid search or an optimization based method requires different approaches to visualization. The data acquired in a grid search lends itself to graphing in contour plot form. Since the data is inherently in a grid, it is relatively easy to take two-dimensional planar slices of the data from the requirements space, and plot the contours associated with each of the system properties technology boundaries. This was the method applied by the author in the comparison of the requirements space for a hypersonic strike-fighter and a hypersonic cruise missile [7]. In the cases where a significant number of system state property limitations are present, there exists a potential for confusion from all of the boundary contours. To alleviate this problem, only the most stringent boundaries need to be included. Furthermore, in the case of the grid-search based method, most of the data collected remains unused by this part of the analysis, but can be saved for use further on in the design process. The method needed for visualization of the data acquired using an optimization based approach, be it a local or global optimization method, is more complicated.

The use of an optimization based approach to discover the locations of the catastrophic systems state boundaries does not inherently produce a grid of data. This problem was discovered by the author during the study of the requirements space

study for a notional high-speed cruise propulsion system [107]. In this case, the data collected best lends itself to a meta-model representation of the catastrophic boundaries. This works for both the individual technology boundaries and the Pareto boundary which is formed by the simultaneous discovery of all the boundaries for a particular system. Once a meta-model is derived, production of any number of Ashby charts involves only the use of the meta-model to predict the boundary at the necessary locations in the requirements hyperspace. Since the boundaries tend to show a nonlinear behavior and a reasonable amount of data is gathered by the EA, the author proposed to use the Gaussian Process meta-modeling technique to capture the catastrophic system state boundaries.

CHAPTER V

IMPLEMENTATION

Due to the highly complex and combinatorial nature of investigating a requirements (hyper)space it is necessary to find a means of simplifying the problem. This task is greatly simplified by the fact that the only points that are of interest are those that lie upon the technology limits. Furthermore, there is no inherent requirement that tremendously large number of points on these boundaries be discovered. Instead it is possible to create a meta-model of the actual boundary surface that can be used in the visualization environment. Because of these simplifications a relatively cheap computational implementation is possible. The implementation consists of two primary parts: the boundary discovery, and the boundary meta-modeling and visualization. Both of these will be discussed in detail in this chapter. A flow chart for the generic computational method is shown in Figure 31

5.1 Process Flow & Setup

While not of the same level of interest as the discovery and modeling of the technology limit derived boundaries that exist in the requirements (hyper)space, the initial problem definition and setup are no less important. Figure 28 on page 48 and Figure 31, show the general and more specific flow of the entire process. This process can be enumerated as follows:

1. Identify Need
2. Define the Problem
3. Identify the Requirements

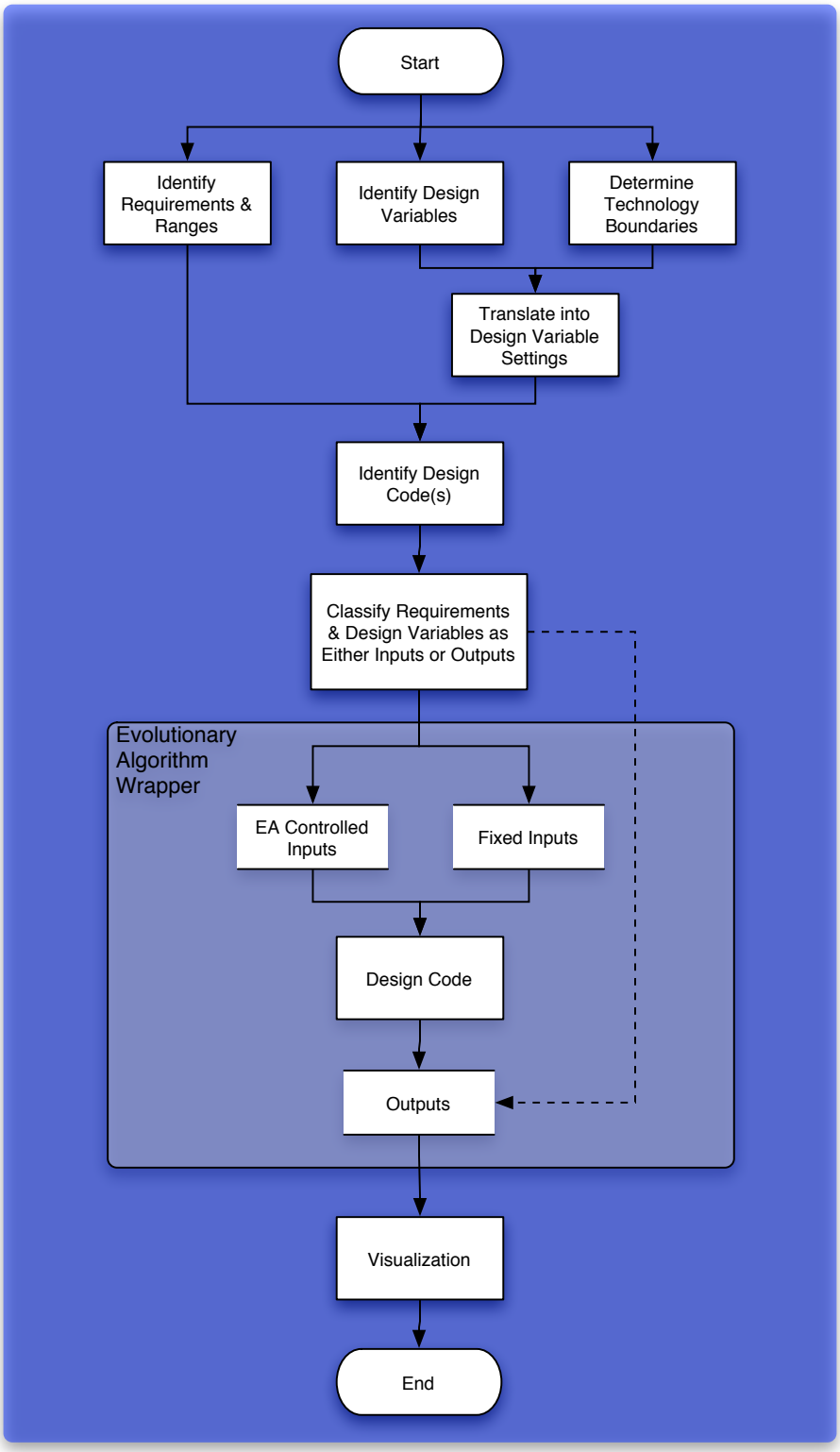


Figure 31: Computational Implementation

- Set Requirement Ranges

4. Identify Potential Systems, Subsystems, and Technologies
    - Identify Subsystem and Technology Limits
5. Identify & Link Computational Tools
    - Translate Requirements and Properties into Variables & Settings
    - Classify Requirements and Properties as Either Inputs or Outputs
6. Operate EA to Discover Boundaries
7. Model & Visualize Boundaries
8. If Necessary Identify new Systems or Technologies
9. Pass Information to Decision Makers

Steps $1-3$, part of 4, and 5 are all part of the current design methodologies. Additionally, the TIES methodology relies heavily upon step 8. However, steps 6 and 7 set RCD apart from the previous methods. The implementation of these steps is detailed in the following sections.

5.2 Boundary Discovery Method Using Evolutionary Algorithms

The key to boundary discovery is identifying a method that is capable of identifying points that lie on or very near to the technology boundary, and ensuring that the widest possible variety of these points be identified. As mentioned in Chapter 4, the multimodal nature of the boundary discovery and the need to ensure as broad as possible a front, make a multi-objective stochastic optimizer ideal for use in discovering the technology boundaries.

The specific class of multi-objective, stochastic optimizers that was chosen as most applicable to the task at hand is the Pareto seeking Evolutionary/Genetic Algorithms (EA/GA). More specifically the Strength Pareto Evolutionary Algorithm (SPEA) was identified as the most promising candidate. Additionally, a second Pareto GA, the Nondominating, Sorting Genetic Algorithm II (NSGA-II) was identified as an alternate method in case the SPEA proved unsuitable. The development of an NSGA-II based approach proved unnecessary as the SPEA proved adequate, with some modifications. These modifications will be detailed in this chapter.

5.2.1 Strength Pareto Evolutionary Algorithm Background

The Strength Pareto Evolutionary Algorithm was developed by Eckart Zitzler and Lothar Thiele of ETH Zürich [91]. The basic premise of the SPEA is to create an external nondominated solution by using a strength based approach.

5.2.1.1 Basic Concepts

While domination and strength may be new terms to the user, their definitions are relatively straightforward. Definition 6 defines dominance.

Definition 6. A point A is said to dominate a point B if and only if all responses $f_i(A)$ are greater than or equal to $f_i(B)$ and at least one of these responses is greater than $f_i(B)$ [91].

$$\begin{gathered} a \succ b \text{ iff} \\ \forall_i \in \{1, 2, \ldots, n\} : f_i(a) \geq f_i(b) \wedge \\ \exists_j \in \{1, 2, \ldots, n\} : f_j(a) > f_j(b) \end{gathered} \tag{26}$$

The strength of a point, which is its fitness, is determined in one of two ways.

1. For the external, nondominated population, the strength is determined as:

Each solution $i \in P'$ is assigned a real value $s_i \in [0, 1)$, called strength; s_i is proportional to the number of population members $j \in P$ for which $i \succeq j$. Let n denote the number of individuals in P that are covered by i and assume N is the size of P. Then s_i is defined as $s_i = \frac{n}{N+1}$. The fitness f_i of i is equal to its strength: $f_i = s_i$ [91].

Where, $i \succeq j$ is i covers j, as shown in Equation 27

$$a \succeq b \text{ if}$$
$$\forall_i \in \{1, 2, \ldots, n\} : f_i(a) \geq f_i(b) \tag{27}$$

2. For the individuals in the internal populations the strength is determined as:

   The fitness of an individual $j \in P$ is calculated by summing the strengths of all external nondominated solutions $i \in P'$ that cover j. We add one to the total in order to guarantee that members of P' have better fitness than members of P (note that fitness is to be minimized, i.e., small fitness values correspond to high reproduction probabilities) $f_j = 1 + \sum_{i, i \succeq j} s_i$, where $f_j \in [1, N)$ [91].

Figure 32 illustrates the concept of strength. Examining the points in Figure 32a, the nondominated points stored in the external population are shown by xs and the internal population members by dots. Looking at the upper-left most nondominated point, there is the fraction $\frac{3}{8}$ next to the value. This is that point's strength. The numerator is calculated by adding up the number of points that the specific individual "covers", in this case that is three. These points are the ones labeled $\frac{11}{8}$, $\frac{16}{8}$, and $\frac{19}{8}$ respectively. The denominator is determined by adding 1 to the number of points in the internal population. In Figure 32a there are seven internal population members; therefore, the denominator is eight. For the internal population members the strength is calculated slightly differently. The strength is calculated by adding the values of

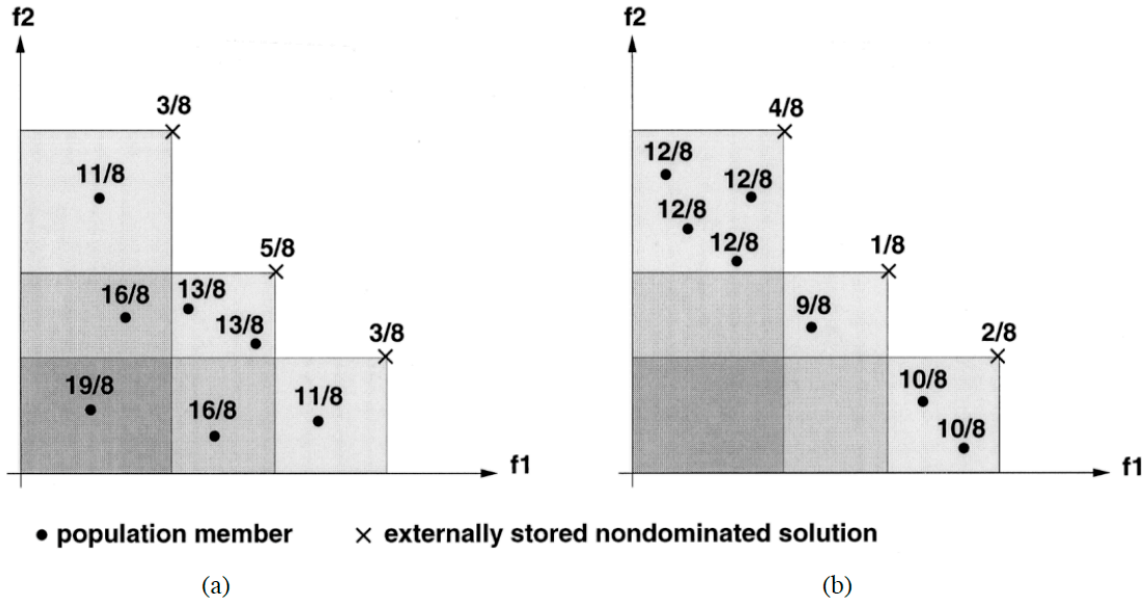


Figure 32: Strength Determination Example [91]

the strength of all external points that "cover" the internal point and adding one. For example, the point in the lower left hand corner of Figure 32a, labeled $\frac{19}{8}$, is covered by all of the external population members; therefore, the strength is given by $\frac{3}{8}+\frac{5}{8}+\frac{3}{8}+1$. This ensures that the internal population members will have a strength greater than one, and the external population members will have a strength of less than one. Therefore, the lower the strength the "better" the fitness for each point. This ensures that those points which are along the Pareto front are preferred.

5.2.1.2 Algorithm

The implementation algorithm for SPEA as proposed by Zitzler and Thiele is very similar to that of other EA/GAs. The only major differences are the addition of the external nondominated population and the calculation of strength as part of the fitness determination process. The basic flow is as follows [91]:

1. Generate an initial population P and create the external, nondominated set P'.

2. Determine domination and copy nondominated points in P to P'.

3. Remove solutions from P' that are covered by other solutions in P'.

4. If the number of solutions in P' exceeds a predetermined limit N' remove points through clustering.

5. Calculate the fitness of each individual in both P' and P using the strength method detailed above.

6. Select individuals from both $P\&P'$ for use in the tournament selection method mating pool.

7. Apply crossover and mutation operators as desired.

8. If the maximum number of generations has been reached, stop; if not loop from Step 2.

In addition to the determination of domination and strength, there is a clustering step for reducing the number of solutions in the external population. This allows a maximization of the coverage of the Pareto front, while maintaining a reasonable number of external points. Clustering identifies a number of clusters, and keeps only the centroids of each cluster. This action is performed until the maximum allowable number of external points is reached. By keeping only the centroids this maximizes the distance between adjacent external points, ensuring a greater spread of points across the Pareto front.

Zitzler and Thiele have made both a program skeleton and an example implementation available on the web [112]. Further, Zitzler and Laumanns have developed an improved version of SPEA known as SPEA2 that improves on the convergence rate of SPEA by modifying the strength calculations [113]. However, since the improvement is mostly in convergence rate, for which SPEA has proved adequate, and there is no example source code available, the original form of SPEA was chosen for continued implementation.

5.2.2 Differences Between Typical Pareto Evolutionary Algorithms and Boundary Discovery

There are three primary differences between the typical Pareto seeking optimization algorithm and the goals of technology boundary discovery. The first is a difference in the what points should be considered nondominated. The second is the need to operate the external function in a reverse manner. The third is the need to let a whole group of state variables, that are inputs to the external code equilibrate as necessary, without attempting to optimize their values.

5.2.2.1 Reformulation of Dominance

Typically a Pareto seeking EA is interested in finding the nondominated points only in the response space. Furthermore, the calculations as to which points are not dominated are relatively straightforward. This is because there is an explicit goal, i.e. minimization, maximization, or closeness to a goal. While this still holds in the technology boundary discovery process, there is an additional requirement: nondomination in the parameter space.

Since the goal of technology boundary discovery is to map the state space response limits into the requirements (control) space, it is necessary to ensure that the points selected are nondominated in both the parameter and response space. This a relatively simple concept, e.g. Equation 26 needs to be modified to refer to the parameter space. If the goal was to maximize distance from the origin then Equation 26 would be modified as shown in Equation 28.

$$\begin{aligned} a \succ b \text{ iff} \\ \forall_i \in \{1, 2, \ldots, n\} : a_i \geq b_i \wedge \\ \exists_j \in \{1, 2, \ldots, n\} : a_j > b_j \end{aligned} \tag{28}$$

Unfortunately, the solution is not that simple. Since there is not inherent preference for distance from the origin, i.e. a point close to $(0, 0)$ is just as valid as one very far

away, the concept of domination needs to further refined.

Since the primary goal of the technology boundary discovery process is to create a close domain in the requirements (hyper)space, the definition of domination should be reformulated to encourage this process, i.e. give preference to points that create a closed surface in the requirements (hyper)space.

5.2.2.2 Need for Capability to Operate External Function in an Inverse Manner

The need to operate external functions in a reverse or inverse manner derives from the dependence of modern design on the use of legacy computational tools. In many of these tools system level requirements are represented as program outputs, and the subsystem & component properties are program inputs. These two conditions require the capability to operate the external computation tool in an inverse manner. To do this the EA must be able to record the values of those requirements that are code outputs, and input those subsystem & component properties that are code inputs.

This is a relatively straightforward task; however, it requires that a separate set of variables be created. Conveniently this variable class is also applicable to the need to allow the semi-independent state variable to equilibrate for each technology boundary point.

5.2.2.3 Equilibration of State Variables

Since the discovery of technology boundaries requires that an equilibrium approach to systems design be followed, it is necessary to allow a significant number of the state variables to vary as needed. Since the typical SPEA implementation will optimize any input variables to find nondominated points the basic implementation of SPEA is insufficient for the needs of equilibrium design processes. To accommodate these needs, a set of variables need to be created that are not directly optimized by the EA.

All of the differences and modifications listed in the above subsections were incorporated into a Modified Strength Pareto Evolutionary Algorithm (MSPEA). This

algorithm will be detailed in the next section.

5.2.3 Modified Strength Pareto Evolutionary Algorithm

The MSPEA was created specifically to address the shortcomings of SPEA when applied to the technology limit boundary discovery process. It modifies the determination of dominance, changes the clustering behavior, adds the capability to run the external function in an inverse manner, and incorporates a set of unoptimized variables that are allowed to equilibrate at each Pareto solution point. The following sections detail the changes made to MSPEA. Further, the program skeleton for MSPEA is given in Appendix D.

5.2.3.1 Modified Domination Determination

Starting from Equation 26, on page 67, which describes how to determine if point a dominates point b in the response space, $a \succ b$, it is possible to create a simple relationship for determining domination in the parameter space. This is shown in Equation 28, on page 71. However, this form is only sufficient if the domination is based upon a smaller or larger is better schema, i.e. one wants to get as far away from, or as close to, a fixed region as possible. This is not the case with the discovery of technology boundaries, at least with respect to the parameter space. In the case of technology boundary discovery, where the combined technology boundaries are transformed into the requirements, control variable, (hyper)space from the state variable space; the desire is to produce a closed domain for each system or system type being investigated. These domains are similar to the notional domains shown in Figures 2 through 8 in Chapter 1.

To achieve the goal of selecting a set of points which are nondominated in the parameter space, in such a way that the closed domains can be formed, a new representation for dominance needs to be created. By modifying Equation 26 to the form

shown in Equation 29,

$$
\begin{aligned}
& a \succ b \text{ iff} \\
& \forall_i \in \{1, 2, \ldots, n\} : g_i(a) \geq g_i(b) \wedge \\
& \exists_j \in \{1, 2, \ldots, n\} : g_j(a) > g_j(b)
\end{aligned} \tag{29}
$$

where,

$$
g_i(x) = \frac{|x_i - x_{i_{nominal}}|}{x_{i_{nominal}}} \tag{30}
$$

This creates a structure where points that are farthest from the $\mathbf{C}_{nominal}$ point are considered dominant. There are, however, two primary problems with this implementation. One, it makes no discrimination over whether or not a point is above or beneath the $\mathbf{C}_{nominal}$ point. Two, the $\mathbf{C}_{nominal}$ point cannot be determined ahead of time.

To solve the problem where a point a is farther away from $\mathbf{C}_{nominal}$ than b, i.e. dominates b, regardless of direction, it was necessary to include a programmatic check on direction, Equation 29 becomes:

$$
\begin{aligned}
& a \succ b \text{ iff} \\
& \forall_i \in \{1, 2, \ldots, n\} : g_i(a) \geq g_i(b) \wedge \\
& \exists_j \in \{1, 2, \ldots, n\} : g_j(a) > g_j(b) \wedge \\
& \forall_i \in \{1, 2, \ldots, n\} : a_i * b_i = |a_i| * |b_i|
\end{aligned} \tag{31}
$$

where $g_i(x)$ maintains the form given in Equation 30.

Addressing the fact that the $\mathbf{C}_{nominal}$ cannot be determined ahead of time is even more straightforward. Since the goal is to provide the greatest possible expanse for the domain, and given the fact that the EA can produce points which are located in significantly different regions of the space during each generation, $\mathbf{C}_{nominal}$ is calculated in each generation, prior to the determination of dominance. Taking

$\mathbf{C}_{nominal} = [x_{1_{nominal}}, x_{2_{nominal}}, \ldots, x_{n_{nominal}}]$, then $x_{i_{nominal}}$ can be calculated using Equation 32.

$$x_{i_{nominal}} = \frac{x_{i_{max}} - x_{i_{min}}}{2} \tag{32}$$

The combination of both $\mathbf{C}_{nominal}$ and the dominance formulation given in Equation 29 to determine the domain span in the parameter space, and the standard response domination form, given in Equation eq:domaindom:mod, allows for a set of nondominated technology boundary points to be selected for inclusion in the MSPEA external population.

5.2.3.2 Clustering Modification

Just as in the determination of dominance, where SPEA functions only upon the responses, clustering in the technology boundary discovery process needs to be modified. Since the spread along the response front is not of particular interest, but spread along the technology boundary in the requirements space is desired, the clustering formulation is modified to cluster in the **C** space and keep only the centroids. Other than changing the variable type upon which the clustering operates, the rest of the clustering routine remains unchanged from that described by Zitzler and Thiele [91].

5.2.3.3 Additional Variables & Inverse Operation

Even though the problems of allowing for reverse operation of the external function/legacy code, and the allowance for variation in nonoptimized state variables are distinct, their solutions are strikingly similar. To address the reverse operation of the external objective function it is necessary to allow for **C**s that are objective function outputs and **Y**s that are objective function inputs. To handle this, two new variable types were created, **T**s and **F**s. Additionally, one of these variable types, **F**, is capable of handling the free-floating state variables.

The first variable type, **T**, are variables that the MSPEA "tracks", i.e. keeps record of, but does not attempt to optimize by. These **T**s are the **C**s that are objective

function outputs. While they are not used as a direct optimization, because they are requirements they are used in the determination of domination and strength. Additionally, like the **C**s that are inputs to the objective function, they are also used during clustering to ensure the maximum possible spread across the technology boundary front.

The second variable type, **F**, are variables that are inputs to the objective function that the MSPEA allows to "float freely." These can be either **Y**s that are inputs to the external function or state variables that are allowed to equilibrate. To ensure that any **Y**s that are inputs are actually used by the MSPEA to determine domination and strength, they should also be included in the **Y** vector. An example input file containing all of these variables and how they are entered is contained in Appendix E.

5.2.3.4 Modified Strength Pareto Evolutionary Algorithm Flow

With the modifications to the determination of domination, clustering, and the variable types that the code handles, it is possible to describe the typical flow for the MSPEA. This flow-chart is shown in Figure 33. Note that the external objective function is only called if the chromosomal bit string changes, i.e. if a crossover or mutation operation occurs on that population member. Since SPEA did not specify the manner in which crossover and mutation were performed, it was necessary to devise methods for both.

Two basic types of crossover operations were implemented in MSPEA. The first is a simple gene swap between two population members. The second is the equivalent of an amino acid recombination. The two crossover techniques are shown in Figures 34 and 35 respectively. To illustrate how each of the crossover operations work, imagine the case of exploration for an fixed-wing vehicle system. The response of each population member is defined by the complete control and state vector, i.e., the

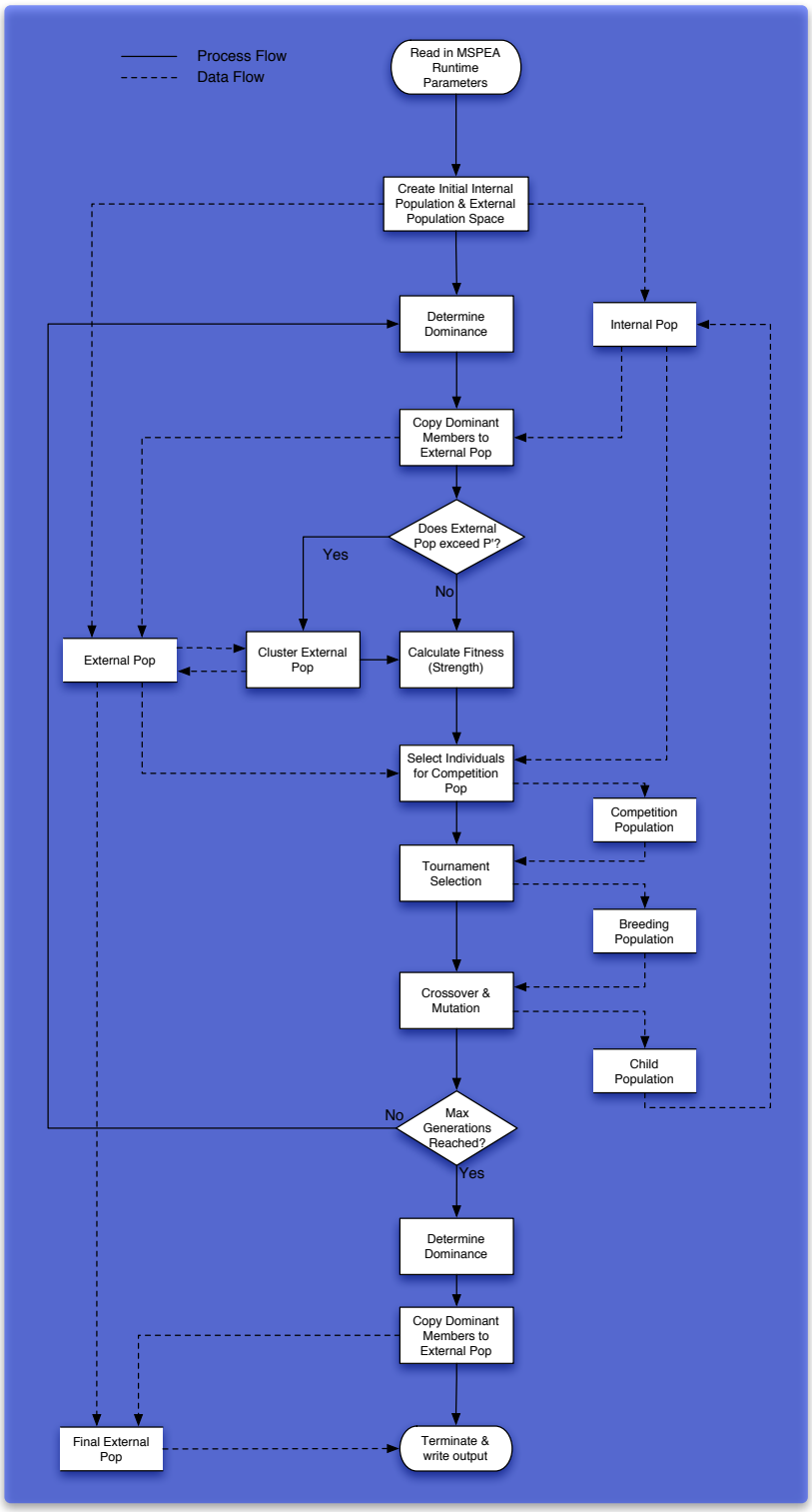


Figure 33: Modified Strength Pareto Evolutionary Algorithm Flow

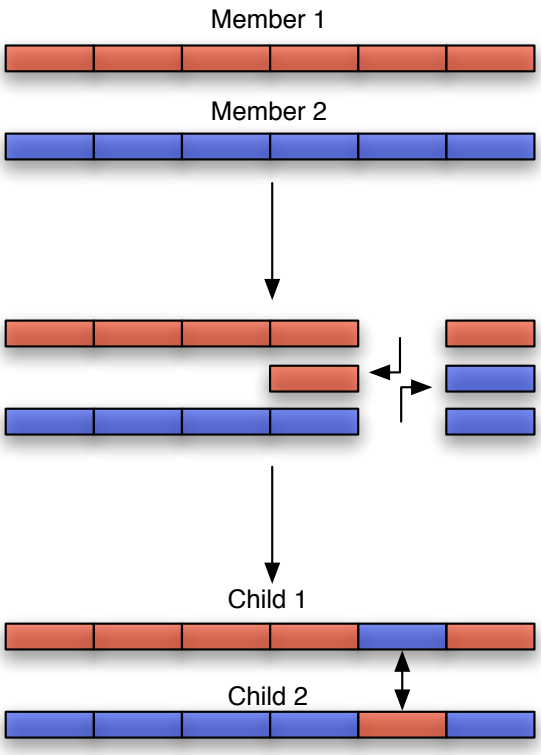


Figure 34: Gene Swap Crossover Technique

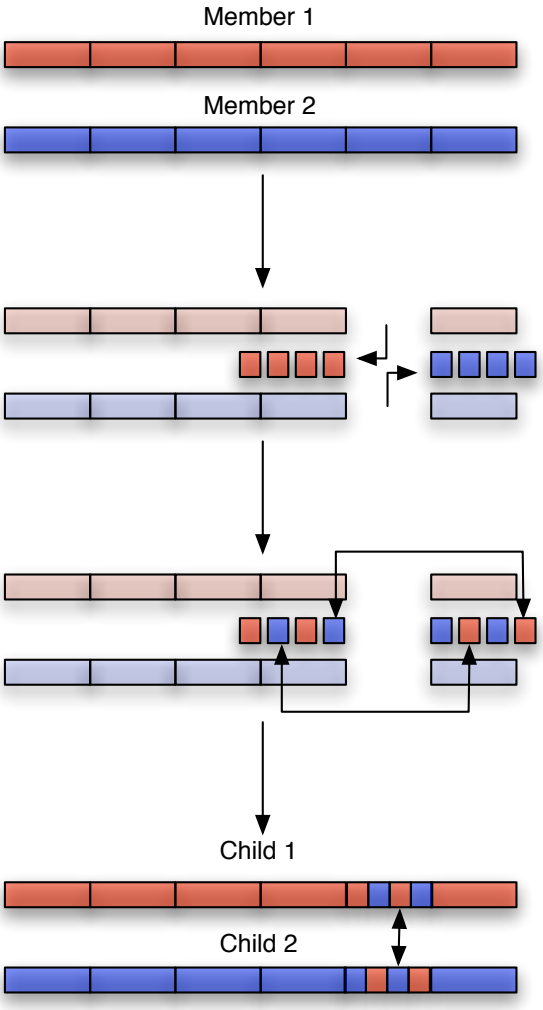


Figure 35: Amino Acid Recombination Crossover Technique

requirements, vehicle attributes, and technologies. Imagine that one of the genes in the member bit string was wing-span. For the gene swap crossover, shown in Figure 34, if member 1 has a wing-span of 50 feet, and member 2 has a wing-span of 45 feet, the crossover would swap values, i.e., child 1 now has a wing-span of 45 feet and vice versa. This mean that child one shares all of the genes of its primary parent, member 1, but a single gene, wing-span, from its other parent, member 2.

The recombination crossover, shown in Figure 35, is less intuitive. The actual value for wing-span is represented, internally, by a discretization, i.e., a specific number in a range, such as $0-1$, $0-3$, $0-17$, etc. Each of these numbers is represented in binary, where 0 in a four-bit string is represented by the binary number $0000b$ and 15 by $1111b$. Using the same wingspan example, suppose that member 1's 50 foot wing-span is represented by the number 9, or $1001b$, and the 45 foot wing span, by the number 7 or $0111b$. The recombination crossover works by randomly swapping bits in a gene; therefore, child 1 might get the gene for wing-span with a value of $0011b$ or 3. Child 2 would then have a wing-span gene with a value of $1101b$ or 13. Note that neither 3 or 13 were present in the parent population members individual genes. Using the same wing span lengths, the number 3 relates to a wing-span of 35 feet, and the number 13 to a wing-span of 60 feet. The recombination crossover, therefore, allows the child population member to jump significantly in the space being investigated by the MSPEA algorithm. The crossover operation, which occurs with a user defined likelihood, is applied to two randomly selected members of the breeding population. Either the gene swap or the amino acid recombination are used, depending on a random choice at runtime. After all members of the breeding population are passed though the crossover step, they are eligible for mutation.

The mutation scheme, which also occurs with a user preset likelihood, is a simple amino acid bit flip, i.e. if the mutation occurs on a bit with a setting of 0, the resulting bit value would be 1. The requisite bit is flipped using the following procedure:

- Randomly choose a gene number from one to number-of-genes
- Randomly choose a bit number from one to number-of-bits$_{\text{gene}}$
- Apply the logical "not" to the chosen bit$_{\text{gene}}$

After the mutation operation is complete the algorithm cycles in the same manner that the SPEA algorithm does.

5.2.3.5 Runtime Properties

Due to the lack of an auto termination capability in both SPEA and MSPEA it was necessary to determine reasonable numbers for the internal and external populations, and the number of generations required. In a simple genetic algorithm that does not store an external, nondominated population, and calls an external objective function for every population member every generation, this is a relatively straightforward task. The user determines which specific combination of population members and generations produces a converged result in the fewest number of function calls. However, because of the presence of both an external population, and external objective function storing, a more detailed study was needed.

The benefit of the response storing method is that it significantly decreases runtime growth as the number of generations increase. This is due to the fact that only a fraction of the data points are re-evaluated at each generations step. This trend can easily be seen in Figure 36. Of interest is that the actual trend, for a 25% crossover rate and a 15% mutation rate, is slightly less than linear. Furthermore, the long term trend indicates that the runtime increases at approximately one-quarter the number of generations.

A somewhat simpler determination is the scaling with respect to the increase in internal (working) population. Since the number of external function calls increases

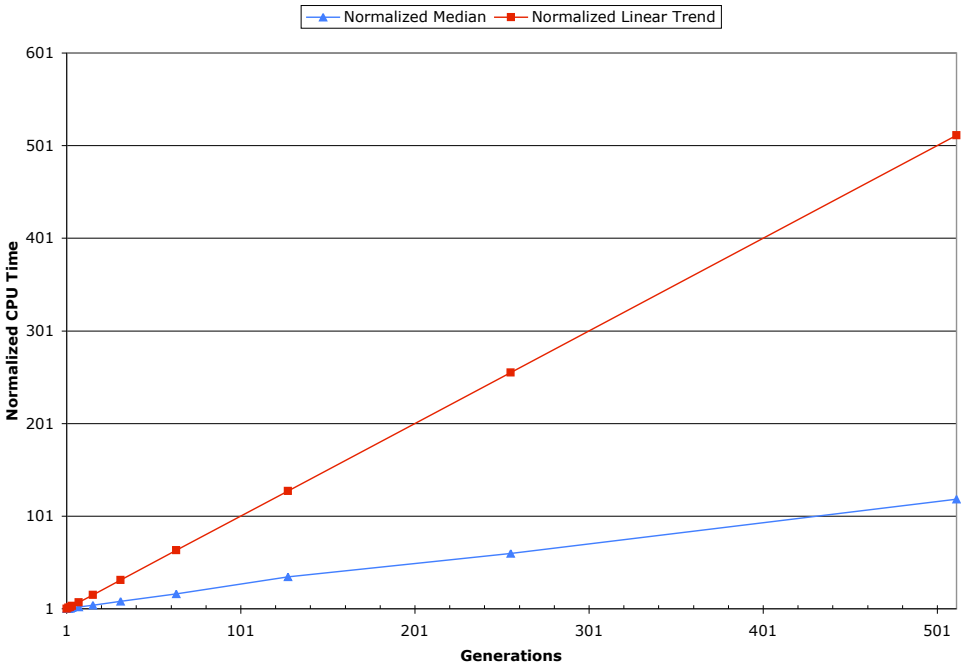


Figure 36: MSPEA Runtime Scaling with Increasing Generations, 50 Internal Population Members

linearly with the internal population, the total runtime should scale accordingly. Figure 37 shows that the runtime actually scales faster than the pure increase in population. This is attributable to the extra overhead that occurs with larger populations, especially with respect to the calculation of domination which scales approximately with the square of the population.

Since the runtime scales at approximately 25% the rate of increase in generations and 115% the rate of increase in internal population members, it would make sense to keep the number of internal population members to a minimum and increase the number of generations as necessary to achieve convergence. The scaling of runtime with an increase in the maximum external population size, shown in Figure 38, which doesn't change appreciably, would seem to support this. However, to determine the proper combination, some knowledge of the number of generations it takes to archive convergence, with respect to the ratio of external to internal population members, is

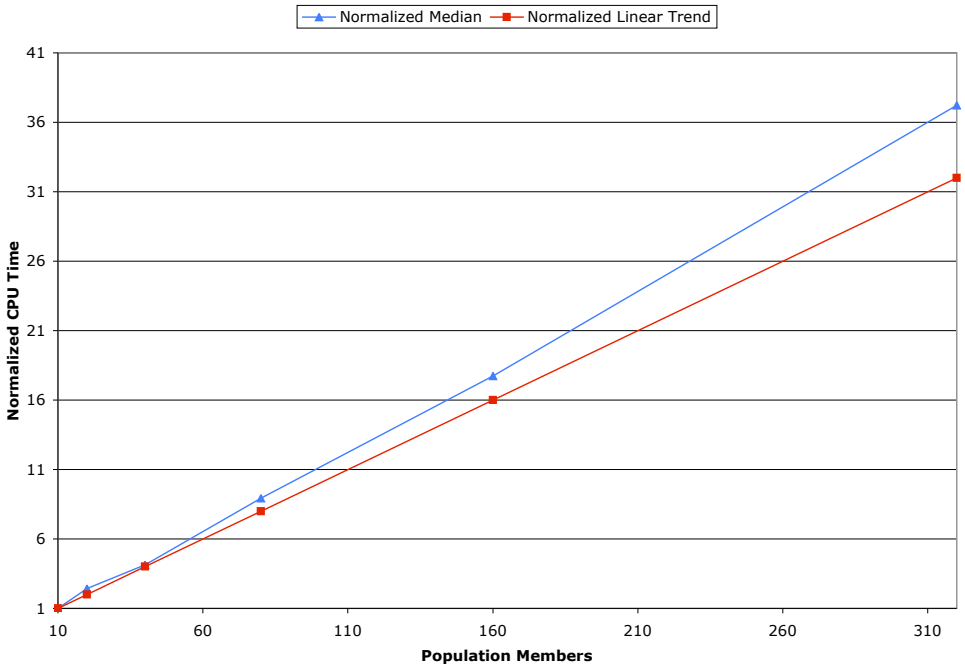


Figure 37: MSPEA Runtime Scaling with Increasing Internal Population, 32 Generations

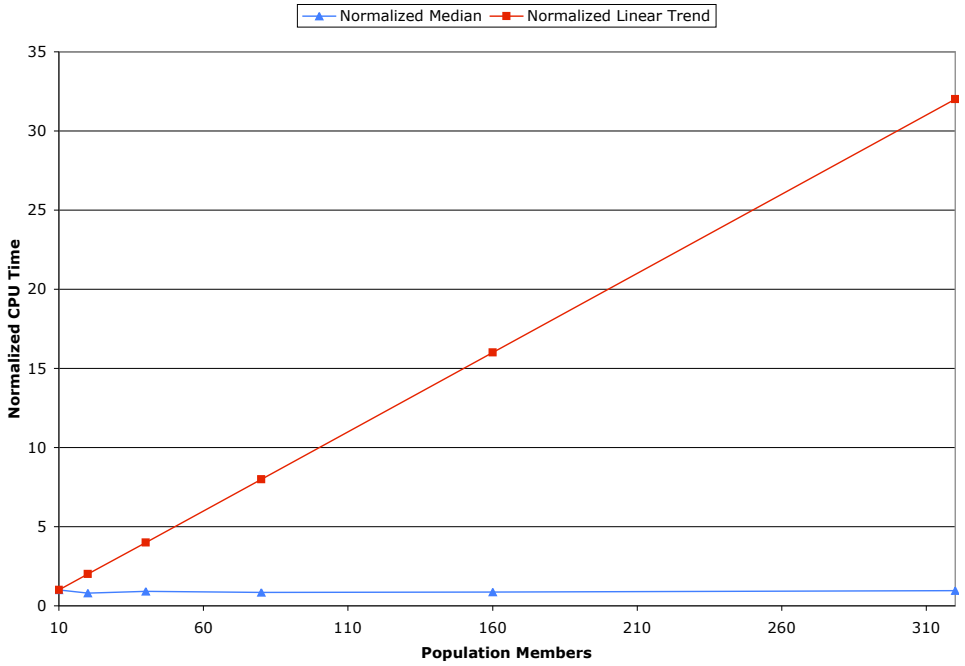


Figure 38: MSPEA Runtime Scaling with Increasing External Population, 32 Generations, 20 Internal Members

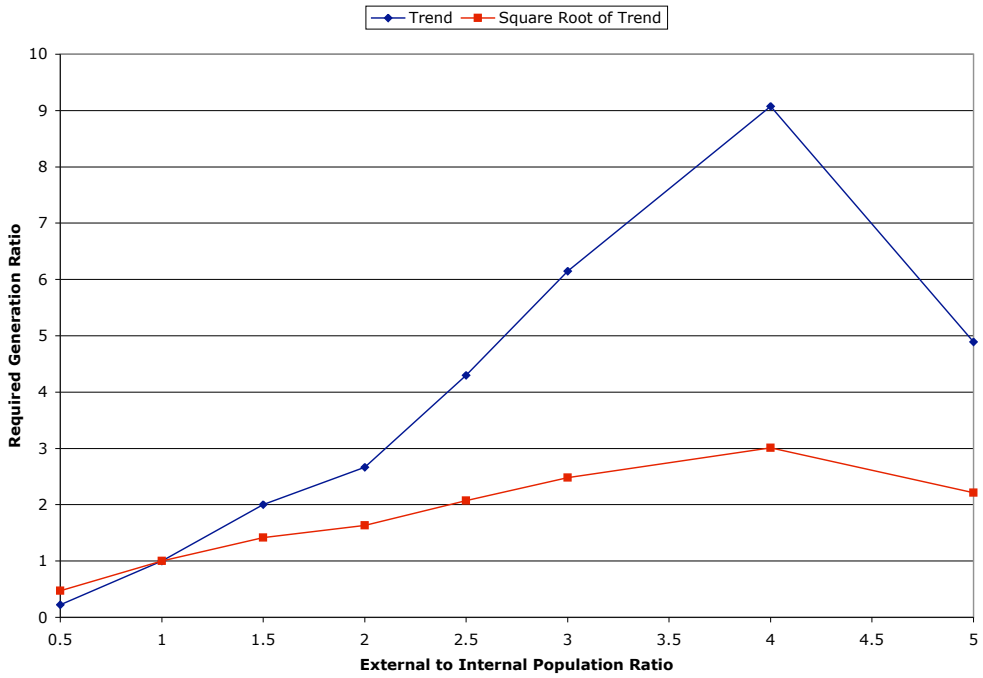


Figure 39: Required Generations Trend with External to Internal Population Ratio

needed. Depending upon the rate at which the number of generations increases with respect to the increase ratio between external and internal population, an "optimum" ratio size of the internal population and the maximum number of generations can be determined for a desired external population size. Figure 39 shows the growth trend in required generations compared to the ratio of external to internal population. Since the required number of generations initially increases with approximately the square of the population ratio, and the significantly faster rate at which runtime grows for increases in internal population, the external to internal population ratio should be set significantly higher than one to one for the most efficient use of MSPEA. The best computational efficiency occurs when the external population is approximately 1.5 to 3 times the size of the internal working population.

5.3 Visualization Method

Because of the highly dimensional nature of the requirements (hyper)space, it was necessary to develop a highly capable visualization environment. Furthermore, unlike the grid search based method described earlier, the boundary seeking methods do not inherently produce an easily dissected surface. Therefore, a means of producing a model of the resulting technology front needed to be devised. By combining a domain slicing algorithm with the meta-model of the technology front, it is possible to view "cross-sectional" slices of the hyper(space) and the technology boundaries that occur on those slices.

5.3.1 Meta-modeling the Technology Boundaries

The MSPEA output, which consists of the external, nondominated population, is typically on the order of 100 to 500 points. While this may seem like a lot of data, in a 15 or 20 dimensional space, these points may be substantially sparse. Further, because of the closed nature of the domain inside the technology boundaries, the points do not directly form a functional model, either in the n-dimensional hyperspace or the two-dimensional cross-sections that will typically be displayed. This fact poses a problem for almost all meta-modeling techniques, as they rely on being able to create a functional representation. There are several possibilities that exist for overcoming this limitation, each of which have their own advantages and disadvantages. The visualization method developed herein creates an artificial response to create a functional, n-dimensional hyper-surface out of the Pareto surface points determined by MSPEA. The creation of this pseudo-functional condition allows for the use of a meta-modeling technique. However, the selection of the proper technique is also critical.

As mentioned in Chapter 4 the most common meta-modeling technique, RSM,

while quick and transparent, is generally insufficient in capability for use in technology boundary discover and modeling. This stems from two properties of the linear regression that is used by RSM. The first is that the model inherently smooths the data, in the case of the technology boundaries the user would like the known data to be faithfully reproduced. Therefore, an interpolative technique would be of more interest. An additional problem with RSM is that it is a linear model, and does not lend itself to highly nonlinear responses which, as shown in Appendices B and C, is a common property for the combined technology boundaries. Therefore, a nonlinear, interpolative meta-modeling technique, such as the Gaussian Process needs to be used for modeling the boundaries.

5.3.1.1 Meta-model Implementation

The creation of the pseudo-functional representation involves importing the results of the MSPEA output, combining the control variable data, and removing any duplicate data points. While the external population filling routine in the MSPEA prevents duplicate data points from entering the external population, by ignoring the "Free" variables there exists the possibility that some duplicate points exist in the requirements (hyper)space. This stems from the fact that a control variable vector, $\mathbf{C}$ does not always produce a unique response. Since the GP does not handle duplicate points well, they have to be removed. This is not a problem as the duplicate points only provide redundant data.

With the duplicate points removed, the artificial functional representation is created. To do this the Pareto data points are assigned a pseudo-Y value of 1. Further, a nominal $\mathbf{C}$ point is calculated as shown in Equation 32. Then, a new set of fully pseudo-data points are created slightly outward from the Pareto points on the radial vector emanating from $\mathbf{C}_{nominal}$. These pseudo-data points are given the pseudo-Y value of -1. Using this data it is possible to train a functional GP.

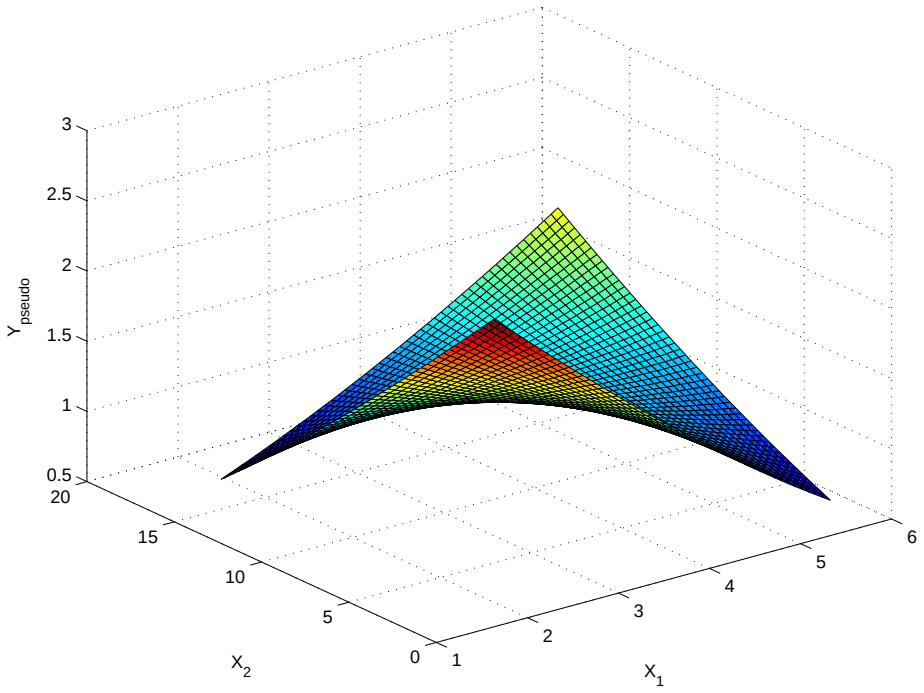


Figure 40: Notional Feasibility Pseudo-Surface

Given that GP training time scales with the cube of the number of data points, that there now exist several hundred real and pseudo data points, and that the GP is Bayesian in nature, the boundary GP is trained with a small, random subset of both the real and pseudo data. By employing the GP training and use flow proposed by Hollingsworth and Mavris [114], it is possible to minimize training time and achieve the maximum fidelity allowed by the known data. The resulting GP produces a surface in the requirements (hyper)space. Any region on this surface which has a pseudo-Y value > 1 is inside the "region of feasibility", meaning that the subsystem level properties are less than their technology limits. If the pseudo-$Y = 1$ then it lies upon the technology boundary, and if pseudo-$Y < 1$ then is lies outside the boundary in the "region of infeasibility." An example of this surface, taken at a slice in a notional requirements (hyper)space is shown in Figure 40.

5.3.2 Visualization Techniques

The benefit of using the pseudo-functional meta-model is that it is inherently suited to direct visualization, especially in the three dimensional form. Using the model of the pseudo-surface as shown in Figure 40 it is possible to produce contours for any two-dimensional slice of the requirements (hyper)space. Unfortunately, given the computational complexity of evaluating a GP and the number of known data points needed to achieve an accurate representation of a highly dimensional surface; real-time visualization requires a relatively coarse contour grid. This stems from the fact that the number of points required to evaluate the pseudo-surface grows with the square of the resolution of the contour chart, i.e. a $10x10$ chart requires 100 points and a $50x50$ chart 2500 points. Therefore each shift in the hidden requirements can take a number of seconds to produce the desired visual result.

5.3.3 Computational Implementation

The visualization environment was implemented in Matlab. It consists of several command line functions all of which fit into one of the following capabilities:

- Visualization model training and improvement
- Feasible region seeking algorithm
- Single and multiple slice plotting routines

The training routine is described above. The primary purpose of the feasible region seeking routine is to find a slice with feasible space, when the initial guess does not provide any regions of feasibility within the region of interest on a particular slice. This is accomplished by "backing" down the vector between the midpoint of the domain described by the known data points, i.e. the results obtained from the MSPEA tool, and the "desired" point on the slice of interest. Once a feasible region appears, the algorithm attempts to bring as many of the requirements settings as

possible to the values desired. The routine is relatively crude but effective. The vast majority of routines in the visualization implementation are provided to produce specific types of visualization.

The plotting routines are designed to form the API basis for any graphical implementation of the toolbox. However, they are currently existent only in command line form. The primary routines consist of a single slice, single system type algorithm, a single system type, multiple slice "sensitivity" chart creation tool, and a single slice, multiple system type plotting routine. Furthermore, because of the computational complexity it is not always possible to produce models that are able to update the display in real-time. Therefore, in order to aid in the understanding of the data, there are an entire set of external routines that produce movies, where a single slice is visualized, with a combination of other requirements varied over time. While this does not live up to the desire for a real-time environment, it does provide a significant amount of understanding.

Since most models will not provide a truly real-time, fully adjustable visualization environment future work in visualization could focus on providing a more optimized visualization capability. This might possibly include the capability to parse regions of the visualization space to multiple clustered machines, similar to the type of routines used in the special effects and digital animation rendering communities. This type of method is appropriate as predictions of the visualization meta-model are only based upon the known data, not predictions at other locations.

CHAPTER VI

VALIDATION

The need to develop a new means of handling system requirements gave rise to the hypotheses to be investigated in this thesis. Research into the validity of those hypotheses and development of the numerical discovery method (Question 2) has produced preliminary results. The classification of requirements and design variables as control and state variables respectively was first demonstrated within the requirements hyperspace for a hypersonic strike system [7]. Additionally, the author demonstrated that the individual systems being studied could be adequately decomposed and modeled using only the subsystem property limits. Furthermore, the use of a grid search method proposed as a possible answer to Question 2 was demonstrated in the same study. However, the limitation inherent to the grid search method necessitated the development of the more capable MSPEA method. The focus of this chapter is to demonstrate the utility of this approach.

6.1 Validation Problem

The ideal validation case for the MSPEA tool is a system, the requirements for which allow for a variety of potential solutions. Given the number of potential rotorcraft system types as shown in Figure 11 in Chapter 1, a rotorcraft based validation case seemed very appropriate. The U.S. Army's Light Helicopter Experimental (LHX) program, which was initiated in 1983, and was originally intended to replace the $7,000$ Vietnam era helicopters in the Army's inventory [115, pp. 2], is a good example of a helicopter program with a diversity of potential solution systems. It was out of this program that the RAH-66 Comanche was developed. Since the original program had

a diverse set of mission requirements which evolved through the 1980s and 1990s it provides an ideal validation case.

6.1.1 LHX Program Background

The U.S. Army started the LHX program in the early 1980s in order to develop a helicopter or family of helicopters that were capable of replacing a host of Vietnam era models, both combat and transport. To facilitate these varied functionalities, the program originally encompassed a broad range of requirements and missions. These included an armed reconnaissance mission, an anti-armor mission, and a utility mission [116, pp. 12]. Each of these primary missions emphasized different attributes of the vehicle. Additionally, there were many other performance requirements for the LHX design. Eventually, due to technical and budget compromises the scope of the LHX project was down sized to a single reconnaissance/attack helicopter [115, pp. 2]. This program eventually evolved into the final RAH-66 configuration seen in Figure 41.

6.1.1.1 LHX Vehicle Requirements

The non-mission profile vehicle requirements for the LHX program are listed in Tables 1, 2, 3, and 4. These tables define general requirements for the LHX vehicle independent of the actual design mission. Since the final system was intended to have a few variants that shared a high degree of commonality, this approach was entirely practical.

6.1.1.2 LHX Mission Requirements

The two principal vehicles were envisioned at program outset, were a combat and a transport configuration. The combat version had two primary missions, an armed reconnaissance mission and an anti-armor mission. The transport aircraft, however, had only a single mission. The two combat mission profiles are given in Tables 5 and

Table 1: LHX Non-mission Requirements [116]

Requirement		**Value**
Deployability	Loading Reassembly	0.5 hr. (1 man hr.) 0.75 hr. (1.5 man hr.)
Reliability	Mean Time Between Failure	$> 100hrs.$
Vibration	Crew Levels	$F = 0.004f + 0.01$
Signature	RCS Infrared Visual	Minimized without penalty Suppressor Weight $<$ 1.5$\times$ Engine Mass Flow Minimize without penalty
Crashworthiness		Design According to US-ARTL TR-79-22
Ballistic Protection	Crew Fuel	7.62mm API (2,600 fps) Self-sealing against 12.7mm
NBC Protection	Crew Avionics	Overpressure, suit allowance EMP/EMI protection
Engines	Type	Existing or advance-technology engine project
Landing Gear	Type Towing capability	Wheel GW across 2.5 CBR soil with quarter ton truck
Anti-Icing	Canopy Engine Inlets Blades	Active Active Passive
Maneuverability		See Tables 2, 3, & 4
Crew	Number Weight	1 250 lb. per man

Table 2: LHX Turn & Roll Requirements [116, pp. 38]

Fwd. Speed (kn)	**Turn Rate (deg/sec)**	**Normal Load Factor (g's)**	**Turn Acceleration**	**Roll Acceleration**	**Conditions**
60	47.7	2.5	-	-	Transient
60	31.5	2.0	-	-	Sustained
150	20.6	3.0	-	-	Transient
150	16.7	2.5	-	-	Sustained
HOGE	60	-	-		Worst-case direction
HOGE	-	-	60 $^\circ/sec$	-	Worst-case direction
150	-	-	-	60° $\triangle$/1.5 sec	All banks angles

Figure 41: Current RAH-66 Comanche Configuration [117]

Table 3: LHX Acceleration Requirements [116, pp. 38]

Axis	**Acceleration (g's)**	**Fwd. Speed (kn)**	**Conditions**
Vertical	1.25	0	Arresting a 1,000 ft/min rate of descent
Longitudinal	0.35 (avg)	HOGE	Zero to 35 knots
Longitudinal	0.40	-	V_{min} power
Longitudinal	0.15	150	-
Longitudinal	-0.30	-	V_{min} power (no gain in altitude)
Longitudinal	-0.30	150	-

Table 4: LHX Symmetrical Conditions [116, pp. 39]

	Enter at V	Attain G	Attain Time (sec)	Maintain G	Maintain Time (sec)	Limit
Pull-up	Best Range	2.0	1.0	2.0	1.0	Airspeed loss to 10 percent Pitch attitude change to 30°
Pushover	Best Range	0.0	1.0	0.0	2.0	Pitch change to 30°
Note:	Pushover performance must also be achievable from exit speed of required symmetrical pull-up maneuver and conversely.					

6. The armament for the anti-armor mission is given in Table 7. The segment profile of the transport/utility version is listed in Table 8. The transport/utility version was also given an additional set of layout requirements to handle the typical payload. These were [116, pp. 44-46]:

- Forward and/or aft facing seats for 95th-percentile troops. Cabin volume to allow for installation of up to eight crashworthy seats.
- In medical evacuation role, carry two litter patients, medical attendant, and crew-chief.
- Internal cabin height not less than 54 inches.
- Cargo doors on both sides, or an aft-loading cargo door, are required. Doors must allow rapid loading and unloading of:
    1. Litter patients
    2. Fully equipped combat troops
    3. GLDS
- Mounting provisions for rescue hoist.
- Mounting provisions for external cargo hook, 3,000 pounds capacity.

Table 5: LHX Armed Reconnaissance Mission [116, pp. 41]

Segment	**Duration (min)**	**Airspeed (kn)**	**Distance**[1] **(nm)**
Warm-up Idle	3	0	0
HOGE Takeoff	1	0	0
Low-level Cruise (Out)	15	$V_{\text{best range}}$	Approx 35
Reconnaissance (Tactical Terrain Flight)	20	$V_{\text{min power}}$	Approx 20
Identify & Assess Enemy (HOGE)	5	0	0
Hand-off to Attack Team (HOGE)	5	0	0
Reconnaissance	15	$V_{\text{min power}}$	Approx 15
Identify & Assess Enemy (HOGE)	5	0	0
Target for Indirect Fire (HOGE)	5	0	0
Reconnaissance	15	$V_{\text{min power}}$	Approx 15
Air Combat Maneuvers	5	120	NA
Low-level Cruise (Back)	20	$V_{\text{best range}}$	Approx 15
HOGE Landing	1	0	0
Reserve Cruise	30	$V_{\text{best range}}$	Approx 70

Notes:

1. Distances are based on estimates for $V_{\text{min power}}$ and $V_{\text{best range}}$ for the 180-knot baseline design.
2. Begin at mission design gross weight
3. Fly entire mission at 4,000 ft & 95°F
4. Compute fuel required at 105% fuel flow rate
5. 700 lb mission equipment
6. 400 lb armament

Table 6: LHX Anti-armor Mission [116, pp. 42]

Segment	**Duration (min)**	**Airspeed (kn)**	**Distance**[1] **(nm)**
Warm-up Idle	3	0	0
HOGE Takeoff	1	0	0
Low-level Cruise (Out)	15	$V_{\text{best range}}$	Approx 35
Tactical Terrain Flight to Battle Area	10	$V_{\text{min power}}$	Approx 10
Maneuver into Attack Position	5	40	NA
	5	0	NA
Attack Air Defense (HOGE)	10	0	0
Maneuver into Attack Position	5	40	NA
	5	0	NA
Attack Armor Targets (HOGE)	20	0	0
Withdrawal in Tactical Terrain Flight	10	$V_{\text{min power}}$	Approx 10
Low-level Cruise (Back)	15	$V_{\text{best range}}$	Approx 15
HOGE Landing	1	0	0
Reserve Cruise	30	$V_{\text{best range}}$	Approx 70

Notes:
1. Distances are based on estimates for $V_{\text{min power}}$ and $V_{\text{best range}}$ for the 180-knot baseline design.
2. Begin at mission design gross weight
3. Fly entire mission at 4,000 ft & 95°F
4. Compute fuel required at 105% fuel flow rate
5. 700 lb mission equipment
6. 700 lb armament

Table 7: LHX Anti-armor Mission Armament [116, pp. 45]

System	**Number of Missiles**	**Total Weight (lb)**	**Total External Draft (ft^2)**
Hypervelocity Rocket	8	500	1.0
Multipurpose Missile	4	200	0.75

Table 8: LHX Utility Mission [116, pp. 45]

Segment	**Duration (min)**	**Airspeed (kn)**	**Distance**[1] **(nm)**
Warm-up Idle	3	0	0
HOGE Takeoff	1	0	0
Low-level Cruise to Pickup	15	$V_{\text{best range}}$	Approx 35
HOGE Landing	1	0	0
Idle	3	0	0
HOGE Takeoff	1	0	0
Low-level Cruise to Forward Area	20	$V_{\text{best range}}$	Approx 45
Tactical Terrain Flight to Landing Zone	20	$V_{\text{min power}}$	Approx 20
HOGE Landing	1	0	0
Idle	3	0	0
HOGE Takeoff	1	0	0
Withdrawal in Tactical Terrain Flight	25	$V_{\text{min power}}$	Approx 25
Low-level Cruise to Base	25	$V_{\text{best range}}$	Approx 60
HOGE Landing	1	0	0
Reserve Cruise	30	$V_{\text{best range}}$	Approx 70

Notes:

1. Distances are based on estimates for $V_{\text{min power}}$ and $V_{\text{best range}}$ for the 180-knot baseline design.
2. Begin at mission design gross weight
3. Fly entire mission at 4,000 ft & 95°F
4. Compute fuel required at 105% fuel flow rate
5. 700 lb mission equipment
6. 1440 lb cargo

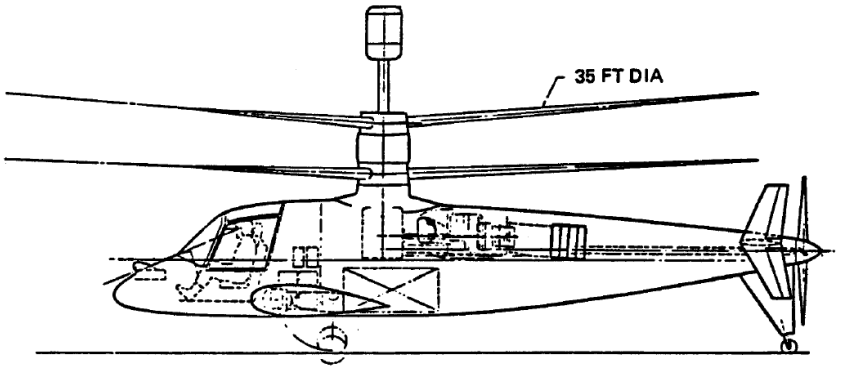


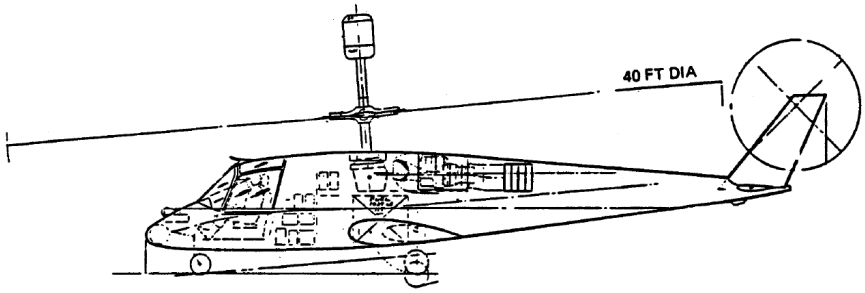
Figure 42: Example Compound Helicopter LHX Configuration [116, pp. 1]

These diverse missions, plus the desire for a high degree of commonality between the different versions put a special pressure on the firms that responded to the Army's request of a new family of light helicopters.

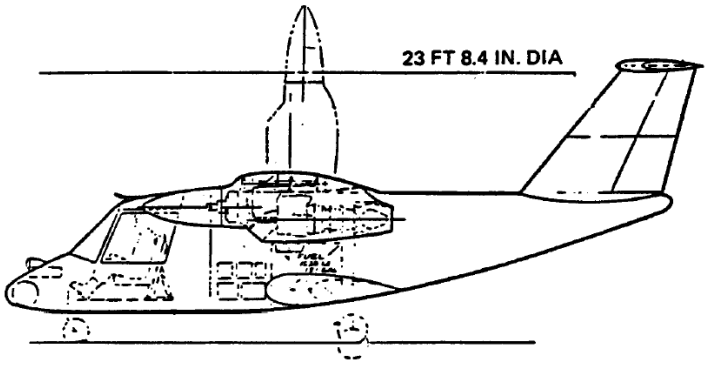
6.1.1.3 Potential LHX Concepts

A variety of different concepts initially investigated for the LHX program. Three basic designs emerged for further investigation: a traditional helicopter, a compound helicopter, and a tiltrotor [116, pp. 3]. Different sub-versions of each primary system were investigated by different manufacturers. However, most settled upon a single main-rotor design for the standard helicopter concept, a coaxial rotor design for the compound, and the wingtip mounted tiltrotor. The primary reason for investigating the compound and tiltrotor concepts was to achieve a higher dash speed capability.

The higher dash speed of the compound helicopter and tiltrotor systems derives from the fact that the lift system is not required to provide forward flight propulsion at high speeds. In the tiltrotor the task of providing lift is transfered to the wings as the vehicle accelerates and the rotor nacelles are rotated into their forward flight position. In the case of the compound helicopter the rotor still provides lift. However, there is a propulsor mounted on the fuselage or at the tail. An example of this configuration is shown in Figure 42. It is this propulsor, plus the problem



(a) Standard Helicopter



(b) Tiltrotor

Figure 43: LHX Concepts [116, pp. 1]

of retreating blade stall that drove the manufacturers to implementing coaxial rotor designs. The coaxial design overcomes the retreating blade stall limitation by using counter-rotating mainrotors. This type of coaxial system, which Sikorsky dubbed the Advancing Blade Concept, allowed for a 215 knot dash speed and a smaller rotor diameter [118, pp. i-ii]. Examples of other LHX concepts are shown in Figure 43.

6.1.2 Current RAH-66 Comanche Properties

The final vehicle which grew out of the LHX program is the Boeing-Sikorsky RAH-66 Comanche. The RAH-66, pictured in Figure 41 on page 93, it is a single main-rotor reconnaissance/light-attack helicopter. A two-view technical drawing for the RAH-66 is shown in Figure 44. The technical specifications for the RAH-66 are listed in Table

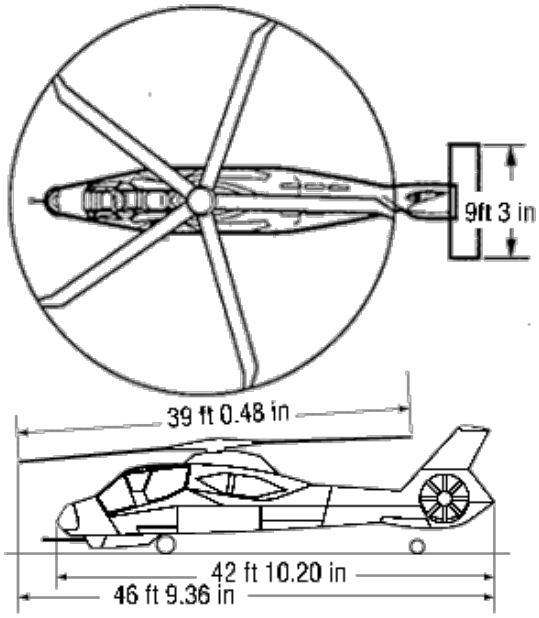


Figure 44: RAH-66 Comanche Two-view Technical Drawing [119]

9. According to Boeing-Sikorsky initial production for the RAH-66 was scheduled to start in November 2004, more than 21 years after the program was initiated [119]; however, in February 2004, the U.S. Army decided to cancel the RAH-66 program.

6.2 Validation Problem Setup

The validation of the MSPEA method, using the LHX example, was designed to demonstrate several features of the RCD boundary discovery method. These include:

- The ability to "quickly" discover and plot sub-system technology limit driven boundaries in the system level requirements hyperspace
- The ability to investigate the effect of changing technology limits on the feasible regions of the requirements hyperspace
- The ability to combine the feasible requirements spaces of multiple potential systems/systems-types to for the requirements Pareto front

Table 9: RAH-66 Comanche Specifications [120, 119]

Specification		**Value**
Length	Total	46.78 ft
	Fuselage	42.85 ft
Width	Rotor	39.04 ft
Height		11.0 ft
Weight	Combat	10,600 lbs
	Empty	7,765 lbs
Powerplant	Number	2
	Type	T800-LHTEC-801 ATE
	Horsepower	1,440 HP/engine
Main Rotor	Type	Bearingless Composite
	Blade Number	5
Tail Rotor	Type	Fantail
Crew	Number	2
Armament		Air-to-air Stinger Hellfire Hydra-70 Rockets 20mm gun
Speed	Dash	∼172 knots (330 km/hr)
	Cruise	∼161 knots (310 km/hr)
Rate of Climb	Vertical	500-850 fpm
Range	Max (internal fuel)	262 nm
	Self Deployment	1,260 nm

- Verify the initial requirements and the track of the requirements, with respect to the RAH-66
  - Verify that the system is capable of meeting its requirements. This serves to validate the usefulness of the MSPEA tool.
  - Determine if the requirements drove technology changes and/or vice versa

As mentioned earlier the LHX program is ideal for this type of validation. It started out as a very ambitious program to replace the U.S. Army's fleet of Vietnam era light helicopters, both combat and utility. Over time through changing needs and budget constraints the program has evolved considerably, to the point where the RAH-66 is a larger, more complex vehicle than was originally envisioned. This can be seen by comparing Tables 1, 5, 6, 7, and 8 to Table 9.

In order to demonstrate the capability of the MSPEA method on the LHX system, a vehicle sizing code needed to be identified, preferably one that could handle multiple types of rotorcraft, especially tiltrotors. The $\mathbf{R_f}$ method is especially useful in this role, as it is a relatively high-level method that can be implemented over a range of fidelity.

6.2.1 $\mathbf{R_f}$ Method Description

The $\mathbf{R_f}$ method, is a relatively simple horsepower and fuel balance method for sizing V/STOL aircraft. Originally a graphical method, it is readily adaptable to computational implementation. A more detailed description of the $\mathbf{R_f}$ method is presented in Appendix F. The $\mathbf{R_f}$ tool used in this study was implemented as a Java library, significantly decreasing the objective function call time.

6.2.2 Validation System Setup

The goals for the validation scheme determined the type and number of different validation test cases that were run. However, before any of these cases could be

Table 10: LHX System Requirements, Inputs to the $\mathbf{R_f}$ Code

Requirement	Variable	Min	Max	Bits	Type
Crew Number	FD.number_crew	1	2	1	Input
Passenger Number	FD.number_pax	0	15	4	Input
Crew Weight	CWD.crew_weight	200	300	4	Input
Passenger Weight	CWD.passenger_weight	150	450	4	Input
Armament Weight	FD.armament_weight	0	2500	4	Input
Sizing Pressure Altitude	MD.pressure_altitude	0	6000	4	Input
Sizing Delta Temperature	MD.delta_temp	0	60	4	Input
Main-Rotor Load Factor	MD.NML	2	5	4	Input
Tail-Rotor Load Factor	MD.NML	2	5	4	Input
Number of Engines	PD.number_engines	1	2	1	Input
Contingency Rating (%)	PD.contingency_rating	25	75	4	Input
Dash Speed	MD.dash_speed	150	300	4	Input
Dive Speed	WD.dive_speed	150	300	4	Input
Download Percentage	MD.download_perc	2.3	4.2	4	Input
Fixed Losses	PD.fixed_losses	10	100	4	Input
Gross Weight	GW_out	-	-	-	Output
Horsepower	HP_out	-	-	-	Output

run the requirements for the LHX listed above needed to be translated into their respective input and output variables for the $\mathbf{R_f}$ library. Additionally, the technology limits, as they were in 1983 and as they are currently needed to be identified and translated into the proper form.

6.2.2.1 System Level Requirements

The LHX requirements set out in Section 6.1.1 needed to be translated into a form that could be used with the $\mathbf{R_f}$ spreadsheet. Some of these requirements translated directly into inputs or outputs in the $\mathbf{R_f}$ library, whereas others needed to be transformed to attain a usable form. The most obvious group was the requirements that translated directly into inputs for the $\mathbf{R_f}$ spreadsheet. Additionally, a range and number of EA bits, for use by the MSPEA had to be proscribed. The non-mission profile requirements that are $\mathbf{R_f}$ inputs are listed in Table 10.

Another group of system level requirements were those that were outputs from the $\mathbf{R_f}$ spreadsheet. Since no input range had to be determined for the MSPEA tool,

the only requirement was that the response values were tracked by the code, and used as part of the determination of dominance and clustering. These requirements and their variable names are given in Table 10.

Since the LHX program was originally intended to produce two vehicles built on a common platform and with a combination of missions, a combined mission profile needed to be created. Some portions were taken from each of the three missions listed in Section 6.1.1. For those portions of the mission that were shared, the minimum and maximum values were chosen to incorporate all of the properties. Conversely, for those mission segments that are only portions of one or two mission profiles, the variable ranges were set with a minimum of zero time; this provides for a case where the segment is nonexistent. The input mission profile requirements are listed in Table 11. The input variables are listed in the *Input* section of the example input file shown in Appendix E. The output requirement variables are listed in the *Tracked* section of the same file.

The reader will note that the description of the requirements shown above, especially the payload and mission profile requirements are relatively detailed and not the simple gross range vs. gross payload that one might expect early on in the design process. The reason for this detail, was an attempt to capture the difference in the two vehicles and three missions that the LHX was originally intended to encompass. Since the missions were provided, the $\mathbf{R_f}$ tool was capable of handling the detail, and a higher level of abstraction was not needed to compare the desired vehicle types, the author decided to use the more detailed description in hopes of obtaining the maximum amount of information from the boundary discovery process. The side effect of this action is that it is no longer a trivial matters to perform simple range vs. payload trade-offs. This was not viewed as a significant detriment; however, in other studies a different combination of abstraction vs. detail might be more appropriate.

Table 11: LHX Mission Profile Requirements, Inputs to the $\mathbf{R_f}$ Code

No.	Segment Property	Variable	Min	Max	Bits
2	Speed	MD.speed[2]	120	160	4
3	Speed	MD.speed[3]	0	100	4
5	Speed	MD.speed[5]	0	100	4
7	Speed	MD.speed[7]	100	140	4
11	Speed	MD.speed[11]	120	160	4
12	Speed	MD.speed[12]	0	100	4
16	Speed	MD.speed[16]	0	100	4
17	Speed	MD.speed[17]	120	160	4
19	Speed	MD.speed[19]	120	160	4
2	Time	MD.time[2]	10	30	4
3	Time	MD.time[3]	0	30	4
4	Time	MD.time[4]	0	20	4
5	Time	MD.time[5]	0	15	4
6	Time	MD.time[6]	0	30	4
7	Time	MD.time[7]	0	10	4
8	Time	MD.time[8]	0	1	1
9	Time	MD.time[9]	0	7	3
10	Time	MD.time[10]	0	1	1
11	Time	MD.time[11]	0	30	4
12	Time	MD.time[12]	0	30	4
13	Time	MD.time[13]	0	1	1
14	Time	MD.time[14]	0	7	3
15	Time	MD.time[15]	0	1	1
16	Time	MD.time[16]	5	30	4
17	Time	MD.time[17]	10	30	4
19	Time	MD.time[19]	20	40	4

6.2.2.2 Subsystem Technology Limits

The other key set of variables are the sub-system and component level driven technology limits. Part of the problem with technology limits is that, for future values, there is a significant amount of prediction and subjectivity involved. For this reason a lot of effort needs to be spent of determining what these values are. However, for the LHX validation case this was not necessary as both the predicted factors and in some cases, the actual values obtained are readily available. Table 12 lists the sub-system driven technology limits for the validation case. The technology driven limits are listed in the *Outputs* section of the example input file in Appendix E.

Additionally, all of the technology limits listed in Table 12 except the main-rotor tip speed are actually inputs into the $\mathbf{R_f}$ spreadsheet. Because of this, the MSPEA tool needed to be able to set values for the sub-system properties. The tool would then drive the properties to their respective limits. The input variables, their respective ranges, and EA bits are shown in Table 13. The input parameters for the subsystem properties are listed in the *Free* section of the example input file in Appendix E.

6.2.2.3 State Variables

In order to create the "equilibrium" method, it is necessary to allow a range of state/design variables to vary within the MSPEA tool. These state variables, which describe various properties of the rotorcraft system are shown in Table 14. The state variables are also listed in the *Free* section of the example input file in Appendix E.

6.2.2.4 Validation Example Systems

In order to get as broad a range of systems as the $\mathbf{R_f}$ spreadsheet can handle, and the maximum exposure of the potential LHX concepts four different rotorcraft systems were used in the validation cases. These systems are:

1. Single-Main-Rotor (SMR)

Table 12: LHX Technology Limits [116, pp. 84, 185]

Limit	**Variable**	**Value**	
		Initial	Current
Contingency SFC	ED.sfc[0]	0.490	0.458 [121]
Maximum HP SFC	ED.sfc[1]	0.490	0.460 [121]
IRP SFC	ED.sfc[2]	0.490	0.464 [121]
Maximum Continuous SFC	ED.sfc[3]	0.501	0.477 [121]
Intermediate Power SFC	ED.sfc[4]	0.531	-
Idle SFC	ED.sfc[5]	0.763	-
IRP Power Fraction(%)	ED.power_fraction[2]	88	89 [121]
Maximum Cont. Power Fraction (%)	ED.power_fraction[3]	73	75 [121]
Intermediate Power Fraction (%)	ED.power_fraction[4]	55	-
Idle Power Fraction (%)	ED.power_fraction[5]	10	-
Maximum HP Time	ED.time[1]	2.5	10 [121]
Contingency HP	ED.hp[0]	1196	1638 [121]
Maximum HP	ED.hp[1]	957	1563 [121]
IRP HP	ED.hp[2]	840	1460 [121]
Max Cont. HP	ED.hp[3]	702	1231 [121]
Intermediate HP	ED.hp[4]	527	-
Idle HP	ED.hp[5]	96	-
Hover SFC	PD.sfc_hover	0.490	0.464
Transmission Efficiency	PD.XMSN_eff	89	-
Main-Rotor Tip Speed	VSD.MR_VTIP	700	-
Bare Engine Wt Factor	TD.bare_engine_K	0.89	-
Main-Rotor Blade Wt Factor	TD.MR_blades_K	0.90	-
Main-Rotor Hub Wt Factor	TD.MR_hub_K	0.80	-
Fuselage Wt Factor	TD_fuselage_K	0.80	-
Wing Wt Factor	TD.wing_K	0.80	-
Tail-Rotor Blade Wt Factor	TD.TR_blades_K	0.85	-
Tail-Rotor Hub Wt Factor	TD.TR_hub_K	0.85	-
Horizontal-Tail Wt Factor	TD.h_tail_K	0.70	-
Vertical-Tail Wt Factor	TD.v_tail_K	0.70	-
Powerplant Section Wt Factor	TD.powerPlant_section_K	0.80	-
Transmission Wt Factor	TD.transmission_K	0.89	-

Table 13: Technology Limit Subsystem Properties, Input Values

Variable	Min	Max	Bits
ED.sfc[0]	0.480	0.510	4
ED.sfc[1]	0.480	0.510	4
ED.sfc[2]	0.480	0.510	4
ED.sfc[3]	0.490	0.530	4
ED.sfc[4]	0.510	0.550	4
ED.sfc[5]	0.720	0.800	4
ED.power_fraction[2]	80	95	3
ED.power_fraction[3]	65	80	3
ED.power_fraction[4]	45	60	3
ED.power_fraction[5]	0	20	3
ED.time[1]	1	5	4
ED.hp[0]	1000	2000	4
ED.hp[1]	800	1800	4
ED.hp[2]	600	1500	4
ED.hp[3]	500	1400	4
ED.hp[4]	300	1000	4
ED.hp[5]	0	400	4
PD.sfc_hover	0.480	0.510	4
PD.XMSN_eff	80	97	4
TD.bare_engine_K	0.5	1.1	5
TD.MR_blades_K	0.5	1.1	5
TD.MR_hub_K	0.5	1.1	5
TD_fuselage_K	0.5	1.1	5
TD.wing_K	0.5	1.1	5
TD.TR_blades_K	0.5	1.1	5
TD.TR_hub_K	0.5	1.1	5
TD.h_tail_K	0.5	1.1	5
TD.v_tail_K	0.5	1.1	5
TD.powerPlant_section_K	0.5	1.1	5
TD.transmission_K	0.5	1.1	5

Table 14: Freely Varying State Variables

Name	Variable	Min	Max	Bits
Main-Rotor Tip Mach	MRD.tip_Mach	0.5	0.9	4
Main-Rotor Solidity	MRD.solidity	0.02	0.12	5
Main-Rotor Blades	MRD.blade_number	2	9	3
Main-Rotor Disc Loading	MRD.disc_loading	5	20	5
Main-Rotor Root Cut-Out (%)	MRD.root_cut_out	0	10	4
Main-Rotor Number	MRD.number	1	2	1
Main-Rotor t/c (%)	MRD.tcMR	5	15	4
Tail-Rotor Solidity	TRD.solidity	0.02	0.12	5
Tail-Rotor Blades	TRD.blade_number	2	9	3
Tail-Rotor Disc Loading	TRD.disc_loading	5	20	5
Tail-Rotor Root Cut-Out (%)	TRD.root_cut_out	0	10	4
Tail-Rotor t/c (%)	TRD.tcTR	5	15	3
Number of Wings	WD.wing_number	0	1	1
Wing Span	WD.wing_span	0	40	4
Wing Area	WD.wing_area	0	250	4
GW on wing	WD.GW_on_wing	0	80	4
Wing Taper Ratio	WD.taper_ratio	0.5	1	4
Wing t/c (%)	WD.tcWING	8	20	4
Wing Incidence (°)	WD.wing_incidence—_angle	-5	10	4
Horiz-Tail Span	HTD.b_HT	5	25	4
Horiz-Tail Pitch Ratio	HTD.P_G	0.05	0.30	4
Horiz-Tail Load Factor	HTD.R_HT	0.1	2	4
Horiz-Tail Area	HTD.s_HT	10	150	4
Horiz-Tail Taper Ratio	HTD.taper_HT	0.75	1.0	3
Horiz-Tail t/c (%)	HTD.tc_HT	5	20	4
Vertical-Tail Span	VTD.b_VT	5	15	4
Vertical-Tail Yaw Ratio	VTD.Y_G	0.05	0.30	4
Vertical-Tail Load Factor	VTD.h_VT	0.1	4	4
Vertical-Tail Area	VTD.s_VT	25	60	4
Vertical-Tail Taper Ratio	VTD.taper_VT	0.5	1.0	3
Vertical-Tail t/c (%)	VTD.tc_VT	5	20	4
De-Icing Equip Wt	FD.de_icing_equipment	10	200	4
Transmission Drive Stages	PD.SKPDSZ	2	3	1
Fuel Tankage Ratio	PD.fuel_tankage_ratio	0.05	0.1	3
Transmission Multiplier	PD.SKT	1	1.3	1

2. Coaxial-Rotor (COAX)

3. Tandem-Rotor (TDM)

4. Tiltrotor (TLTR)

Due to the limitations of the MSPEA tool and the $\mathbf{R_f}$ spreadsheet, each of the vehicle types needs to be run in the MSPEA tool independently. This means that any requirements Pareto front needs to be built after-the-fact, during the post processing of the MSPEA results.

Additionally, while the $\mathbf{R_f}$ spreadsheet/tool cannot yet size a compound helicopter, it can size all of the systems listed above that were ultimately chosen for the RAH-66 Comanche. Further, the capability of analyzing a tiltrotor will allow differences in requirements feasibility between multiple classes of V/STOL aircraft.

6.2.3 Validation Test Cases

In order to verify the applicability of the MSPEA tool and investigate the feasible requirements (hyper)space for the LHX, several different test cases need to be run. These test cases fall into two primary categories: individual system cases and the requirements Pareto front cases. As stated above, limitations in the $\mathbf{R_f}$ spreadsheet and the MSPEA tool meant that the requirements Pareto front could not be generated directly. However, it could be generated by combining the results of all of the rotorcraft systems for the combined technology boundary results.

Since the fully combined boundary case is of the most interest and demonstrates the capability of the MSPEA tool to analyze any smaller combination of boundaries, it was best of the possible technology boundary cases to investigate. However, in order to map the changes in requirements that took place through the course of the LHX program, different combinations of technology levels were investigated, including:

- Basic, 1983 LHX Technology levels

Table 15: Total Hyperspace Resolution Compared to Points Discovered

Total Bits	Potential Points	MSPEA External Pop	Percentage
144	$2.23E+52$	600	$2.69E-39\%$

- Current RAH-66 Comanche technology levels

These cases allow a depiction of the feasible space for the LHX program and an understanding of the actual effect of a change in a single or multiple technology boundaries. Furthermore, because the technology boundary changed may or may not be the most stringent, there is no guarantee that a change in one boundary will effect the feasible requirements (hyper)space.

6.3 Validation Results

Several different results can be obtained from the validation cases. In addition to the prediction of the region of feasibility in the requirements hyperspace, it is also possible to investigate the ability of the visualization scheme to quickly produce the required information.

6.3.1 Visualization of Highly Dimensional Hyperspaces

One of the first problems identified with the visualization of the validation cases, was the dimensionality of the requirements hyperspace for the LHX program. At 43 dimensions, the percentage of the hyperspace that the technology imposed feasibility boundary will occupy is exceedingly small. Furthermore, because of the nature of the MSPEA, i.e. only the technology driven boundary is actually sought, the number of points discovered on this boundary is even smaller as a percentage of the total space. Table 15 gives the total number of points allowed by the discretization and the percentage of this space that the results of the MSPEA algorithm encompass. In addition to the potential that the MSPEA tool may miss a feasible point, the low ratio between the MSPEA results and the total size of the space poses a problem

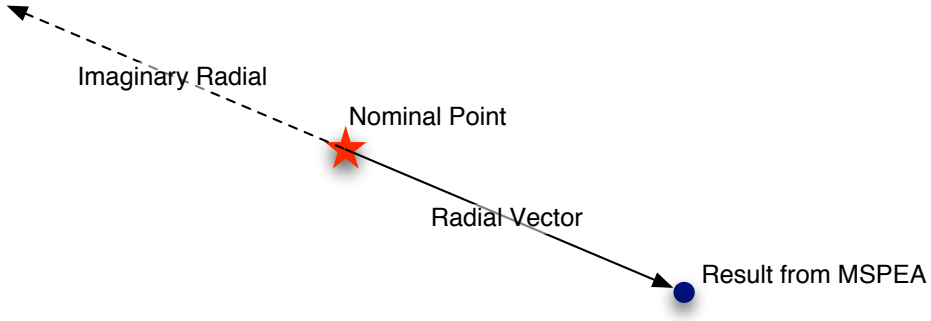


Figure 45: Cross-Sectional Radial, Notional Representation

when creating a visual model.

The primary problem is ensuring that the Gaussian process accurately depicts the values of the pseudo-function within the feasible space. The visualization technique described in Chapter 5 relies on pseudo-functional values to produce the feasibility space contour. To achieve this, additional interior and exterior points are added with either higher or lower pseudo-functional values, depending on whether or not the points are within the feasible portion of the hyperspace. However, because of the relatively sparse nature of these points, the Gaussian process may not accurately capture the pseudo-function. To ensure that it does, a cross-sectional plot of the pseudo-function value on a radius outward from the nominal point to a known boundary point was investigated. A notional depiction of this vector is shown in Figure 45. Additionally, to understand the remaining behavior of the model it is possible to mirror the radial about the nominal point, thereby investigating the prediction of the model in a direction for which no known data point exists. This is also depicted in Figure 45.

Further, to ensure that the Gaussian process model does not accidentally report a feasible result in the infeasible space the pseudo-function values can be predicted beyond the known point. The ideal results, based upon the convention used in the

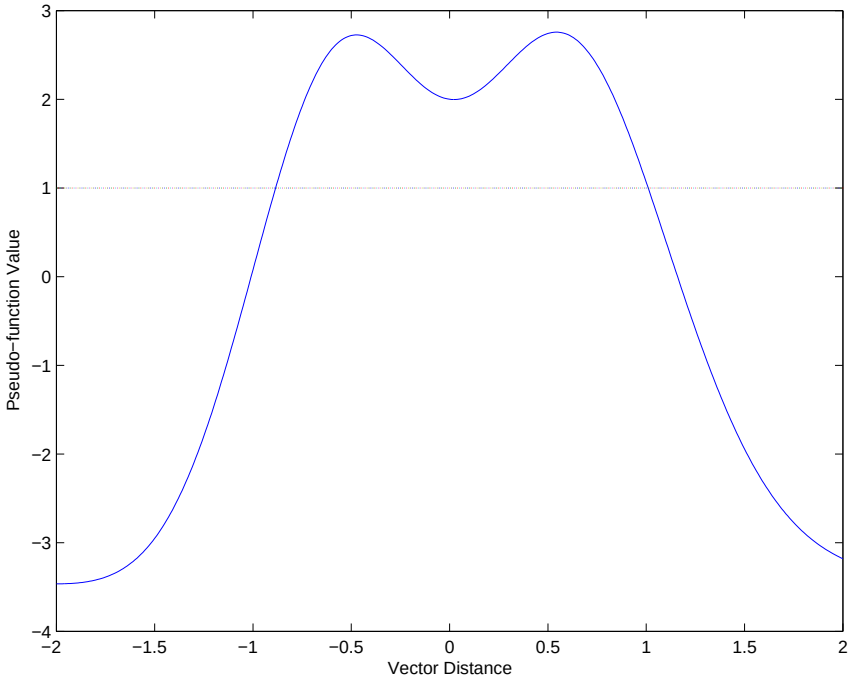


Figure 46: Coaxial Rotor Model Radial Cross Section

visualization model, would be values significantly greater than one within the feasible space, and values significantly less than one outside of the feasible space. This operation was performed for each of the system types, and examples are shown for the 1983 LHX combined technology boundary case.

6.3.1.1 Coaxial Helicopter Feasibility Radial Cross Section

The coaxial main rotor, 1983 LHX combined technology limit meta-model incorporates approximately 330 unique points obtained from the MSPEA code. These are augmented with several pseudo points both inside and outside of the boundary along radials between the nominal center point and the known boundary points. The resulting meta-model predictions along a representative radial is shown in Figure 46.

Reading Figure 46 is not particularly straightforward. The 0 point on the *Vector Distance* axis is the nominal or center point of the feasible domain and the +1 location

is the known boundary point obtained from the MSPEA external population. A properly trained meta-model will produce a pseudo-functional value of less than one for point > 1 on the *vector distance* axis, a pseudo-functional value of one at the $+1$ point, and a pseudo-functional value greater one in the region $0 > x > 1$ on the *vector distance* axis. Any values on the negative portion of the *vector distance* axis are predictions of the visualization meta-model in a direction for which no actual results from the MSPEA code exists.

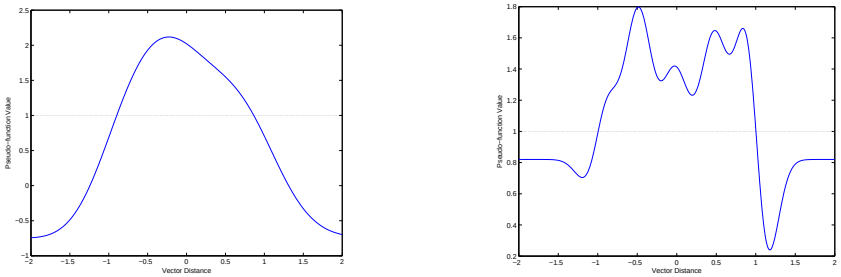
6.3.1.2 Other Vehicle Type, Meta-model Feasibility Radial Cross Sections

The verification of the visualization meta-models for the three remaining vehicle types is substantially similar to that shown for the COAX vehicle type above. Because of this the results have been placed in Appendix G for brevity.

While the performance of each vehicle types meta-model on a cross-section taken from the MSPEA results for that vehicle type are instructive with respect to the accuracy of the meta-model, it is of general interest to understand how the meta-models perform on cross-sections that are not taken from their known data. This ensures that the model produces viable predictions on vectors which contain no MSPEA boundary results.

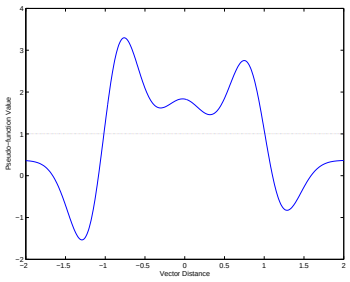
6.3.1.3 Example of Visualization Models on Other Cross Sections

As an example of the investigation of the performance of the system type visualization meta-models on cross sectional radii not taken from their MSPEA response data the SMR, TDM, and TLTR meta-models were run on the cross section for the COAX 1983 LHX combined visualization meta-model. The results of these are given in Figure 47. All three meta-models, when used to predict the pseudo-function values for their respective vehicle types on the vector used for the COAX vehicle type shown in Figure 46, show both feasible and infeasible regions.



(a) SMR Model

(b) TDM Model



(c) TLTR Model

Figure 47: System Type Model Predictions from COAX Model Cross Section Radius

6.3.1.4 Other Comments

Unfortunately, because of the relatively high dimensionality of the requirements (hyper)space, the number of points that must be included in the Gaussian process to produce meta-models that accurately depict the space is relatively high, on the order of 500 to 2000 points. Since the computational requirements for a Gaussian process increase with the square of the number of points included in the meta-model, the runtime to produce visualization meta-model predictions is comparatively high.

This is, unfortunately, a side effect of the use of a Gaussian process in a highly dimensional space, with highly nonlinear data. That is, the pseudo-function is inherently nonlinear with respect to the surrounding requirements (hyper)space. Therefore, in order to accurately meta-model the space, a relatively large number of points need to be included in the model. As the number of dimensions grow, the number of needed points grows. Unfortunately, if the engineer wants to investigate a large number of requirements the best solution to this problem may involve waiting for improvements in computational power.

6.3.2 Combined Technology Boundary Results

Two basic combined technology boundary conditions exist from the historical LHX data: the original 1983 technology limit boundaries, and the current boundaries for the RAH-66 Comanche. Additionally, several of the requirements for the vehicle have been updated over the course of the program. These requirements were given in Tables 1, 2, 3, 4, 5, 6, 8, and 9. The technology boundaries for both cases are listed in Table 12.

Since the primary purpose of the combined technology boundary test cases is to demonstrate the capability of the method in discovering the feasible region of the requirements (hyper)space for the LHX and preferably to validate that the system type chosen was the proper choice, each feasibility contour chart also needs to show

the location of the appropriate LHX requirements.

6.3.3 Original 1983 Technology Limits

Any investigation/validation of the original 1983 technology limits requires that the model be setup to use the same assumptions as the US Army and its contractors used. To achieve this the calibration settings of the $\mathbf{R_f}$ tool were adjusted to match the conditions given in the LHX documents [116, 118]. With this accomplished the MSPEA tool was run using the settings given in Tables 10, 11, 13, and 14, for each of the four vehicle types. The results obtained were then used to create the feasibility pseudo-function and the feasibility space meta-model and to produce the visualizations.

6.3.3.1 Single Main Rotor Vehicle Type

Since the RAH-66, the final product of the LHX program is a SMR helicopter it is most appropriate to investigate the feasibility space for the SMR type first. Furthermore, for the purposes of validation the feasibility region of the SMR system type should include the LHX requirements. The feasibility region taken at the LHX system and reconnaissance mission requirements is shown in Figure 48. The red "X" indicates the original LHX combat helicopter requirements, 700 lbs. of armament and $1,440$ horsepower at a gross weight of $7,713$ lbs. Of interest is that at this particular slice the armament weight can be increased significantly without effecting the horsepower limit. The reason for this is that given the technology boundary combination and the settings of all the other requirements investigated, the armament weight has very little effect on the horsepower.

It is also possible to view the LHX in other planes. Given the fact that the number of crew members has changed over the course of the LHX program, the effect of changing the number and weight of the crew and the armament weight are of interest. These are shown in Figures 49 and 50. Again the red "X" represents the

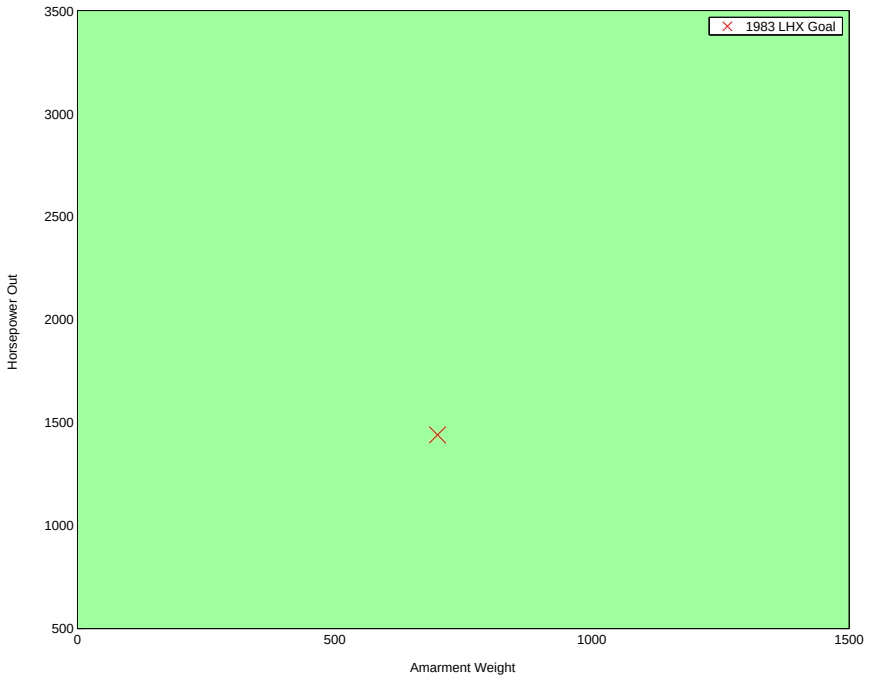


Figure 48: SMR Combined Technology Boundary Feasibility Region, Armament Weight vs. Horsepower

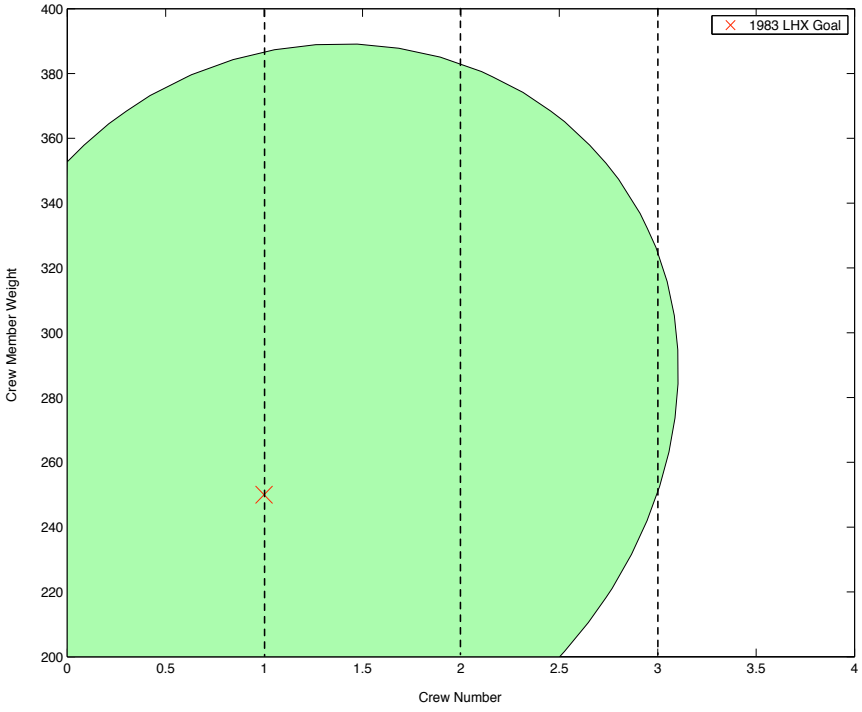


Figure 49: SMR Combined Technology Boundary Feasibility Region, Number of Crew vs. Crew Weight, 1983 Settings

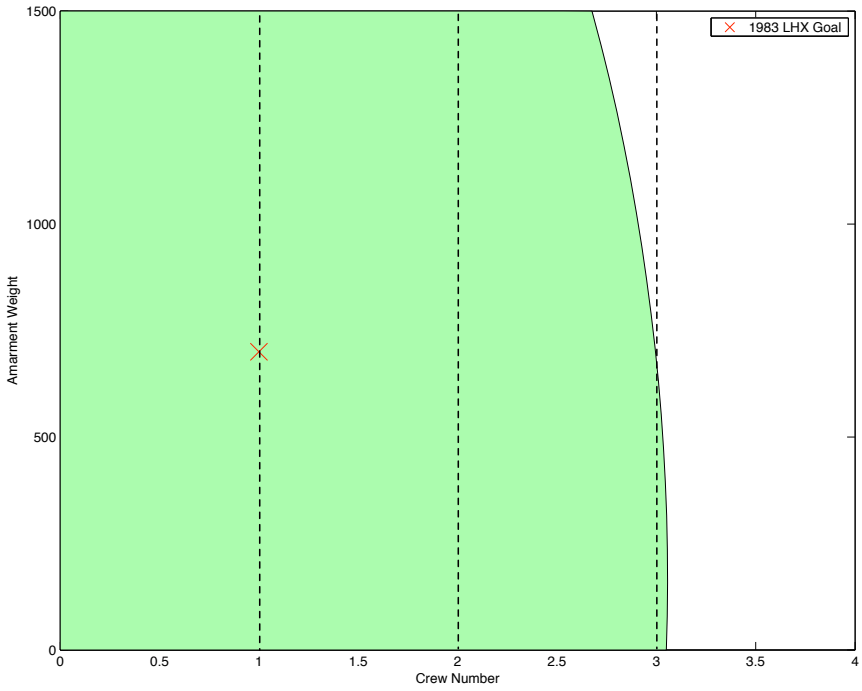


Figure 50: SMR Combined Technology Boundary Feasibility Region, Number of Crew vs. Armament Weight, 1983 Settings

1983 LHX program goal for the combat helicopter.

The reader should note that the number of crew members is typically considered a discrete variable. Why then would the results in Figures 49 and 50 show feasibility boundary that is continuous with respect to the number of crew? There are several reasons for this. The simplest is that the MSPEA tool and the visualization tool do possess knowledge of the continuous/discrete nature of the variables they operate on. In the case of MSPEA all of the variables are inherently discrete because of the EA nature of the method. The visualization routine, and the Gaussian process method it is based upon, treat all input parameters as inherently continuous. Therefore, whereas the MSPEA could only select between one, two, etc. crew members, the visualization routine does not have the capability of discretizing its results in such a manner. Further, when there is uncertainty as to the actual location of the technology boundaries, quantitative or qualitative, it is inappropriate to show only single,

"deterministic" points on the boundary. In this study the inputs were deterministic; however, since they represented the predictions from 1983 of what would be possible in 1990-1993 there is an inherent qualitative uncertainty as to their settings. Neither of the tools pretends to be more intelligent than the user, and a sloppy user runs the risk of producing poor results. Ideally, a finalized, production version of the visualization scheme would contain the ability to classify parameters as continuous or discrete, thereby limiting the values that can be selected during the visual investigation of the feasible space for different settings of the requirements.

Figure 49, is potentially the most interesting of the group, as it shows both the front-side and the back-side of the requirements feasibility region. The back-side of the feasibility region occurs where the settings of the other requirements actually cause the "marginal rate of transformation" to become negative. That is where a less stringent value of one requirement requires a more stringent setting of another requirement. This is one of the primary aspects that separates RCD from other Pareto front based methodologies. In most cases one only has to worry about the positive side of the marginal rate of transformation. However, with system requirements it is actually possible to place oneself "on the back-side of the power curve." These conditions are driven by the implicit lack of preference for a setting or direction of improvement in any of the requirements. This means that the settings for any one of the 41 other requirements are taken as "absolute", that is any deviation off of the proscribed values is unacceptable. This simplifies the exploration, and allows preference for requirements and their specific settings to be added later in the process which, in reality, these preferences will be applied, potentially alleviating the "back-side" limit.

Given the highly complex relationship between the requirements settings and the feasible region it is generally helpful to obtain an understanding of the sensitivity of the feasible space to changes in different requirements. This can easily be done in a

"frozen" manner, i.e. all of the requirements for which the sensitivity is not being studied are kept fixed. Obviously it is not practical to show all of the sensitivities for all 43 of the requirements in this document; this would amount to over 900 individual charts. However, a representative mix of different requirements are shown in Figure 51. While the results of the sensitivity charts for the SMR case of the 1983 LHX requirements is primarily of example value, these charts can, potentially, be used to help determine what changes in requirements need to be made to bring an unfeasible setting in the requirements toward a feasible one.

Also of potential interest are the vehicle types that were eliminated during the design of the RAH-66. These may or may not possess regions of feasibility in planes cut through the requirements (hyper)space about the specific LHX requirements point. Additionally, if these feasibility regions exist they may not cover the specific LHX requirements point. It is, therefore, possible to investigate the sensitivity of the requirements feasibility region to settings of different vehicle and mission requirements.

6.3.3.2 Additional Vehicle Types

The results for the remaining single vehicle type explorations have been placed in Appendix G as they do not, individually, provide any significant new understanding about the problem.

6.3.3.3 Multiple Vehicle Type Visualization

Since all four vehicle types were capable of meeting the basic 1983 LHX requirements, it was easy to produce the multi-vehicle comparison. Figure 52 shows the results of overlaying the feasible regions for multiple system types about the LHX requirements. From this it is relatively easy to grasp the different behavior of the TLTR system as compared to the more standard rotorcraft. By replacing the filled in feasibility regions with the boundary contours, the whole of the feasible region for each vehicle type becomes more evident. This is shown in Figure 53

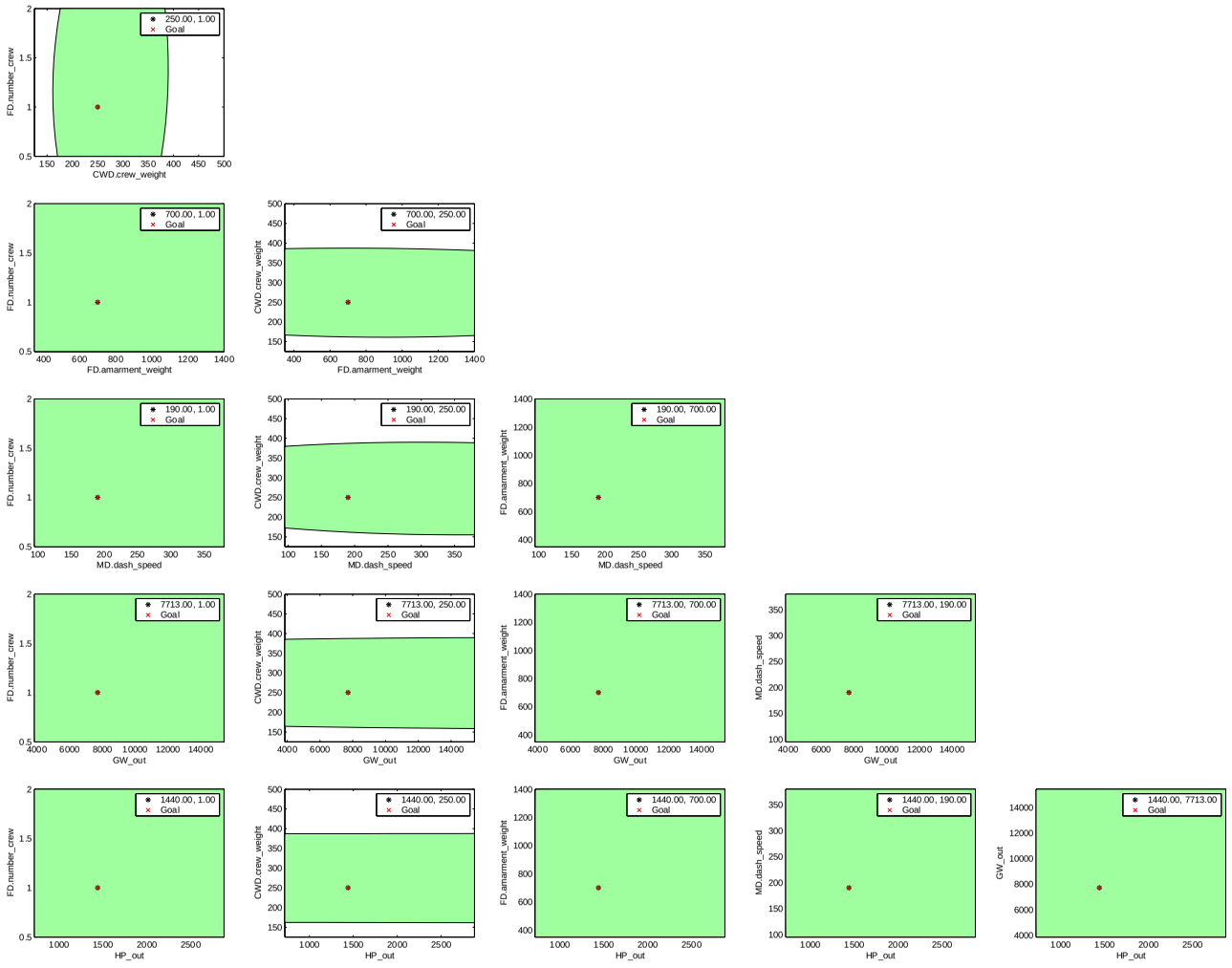


Figure 51: SMR Requirements Feasibility Region Sensitivity, 1983 Settings

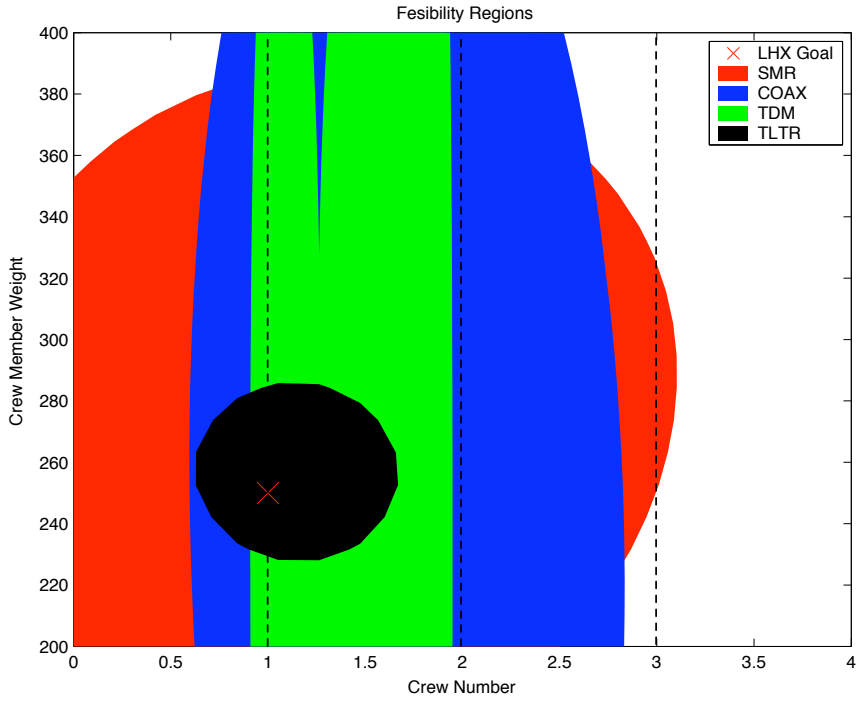


Figure 52: Multiple Vehicle Type Feasible Requirements Regions

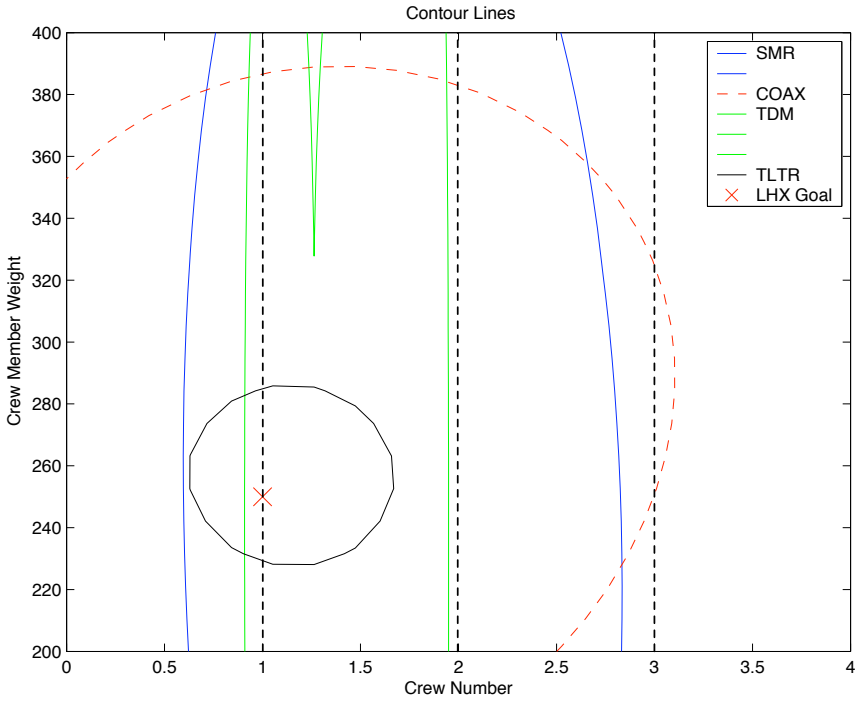
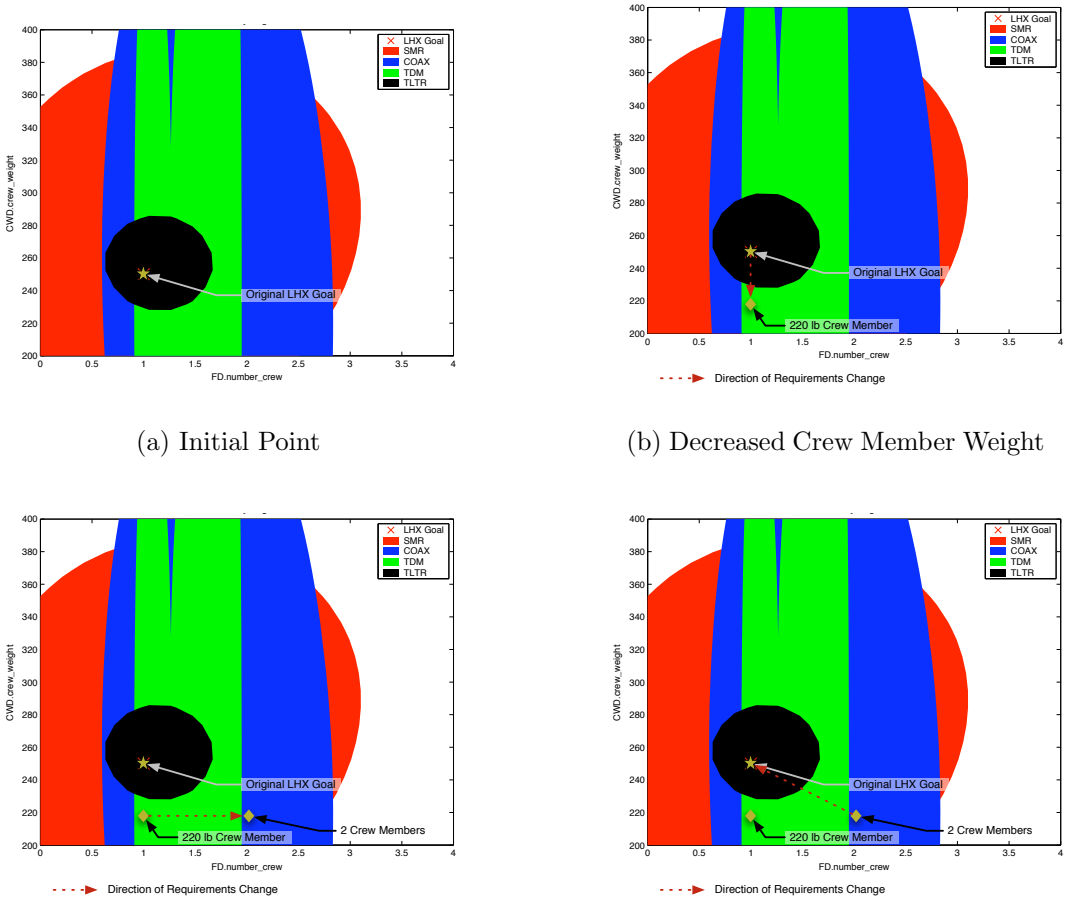


Figure 53: Multiple Vehicle Type Feasible Requirements Boundary Contours

Since all of the systems are capable of meeting the original LHX requirements at the technology levels prescribed earlier in this chapter, it is not directly possible to eliminate any specific vehicle type. History shows us, however, that some or all of the vehicle types were not capable of meeting either the original or changing requirements. This infeasibility could have arisen from either requirements that were not considered in this analysis, such as cost, signatures, etc. or the inability of the technologies to meet the values promised. Ideally nondeterministic technology limits would be used at the beginning of a new program. However, this was not possible in this study because none of this information was available in the LHX documentation cited herein [115, 116, 118, 119, 120].

Therefore, it is of interest to illustrate what potentially might happen to the available system choices in the face of a changing set of requirements. Comparing the SMR to the TDM vehicle systems, each of which occupy different domains in the crew number vs. crew weight regional slice of the requirements (hyper)space, it immediately becomes evident that all systems are capable of fulfilling the baseline single crew member, who weighs 250 lbs; however, by changing either the number of crew or the weight of the crew members it is possible to exit a systems feasible space. This example is shown in Figure 54. For example, if the TLTR type is chosen and the crew member weight is decreased from 250 to $\sim$ 220 lbs, Figure 54(b), the TLTR system is no longer technologically feasible. This stems from the fact that all of the the other requirements are treated as absolute, i.e., the value are not allowed to vary from there settings. In some cases by changing some of background requirements the feasible space might open up. I other cases the positions of the technology limits may not allow a larger space. In this case the design would either need to switch to one of the other system types, or potentially a new technology would have to be identified to open the feasible region for the SMR. If the TDM vehicle type was now chosen, and the number of crew members was increased, shown in Figure 54(c), the designer



(a) Initial Point

(b) Decreased Crew Member Weight

(c) Increased number of Crew

(d) Return to Original Point

Figure 54: System Type Model Predictions from COAX Model Cross Section Radius

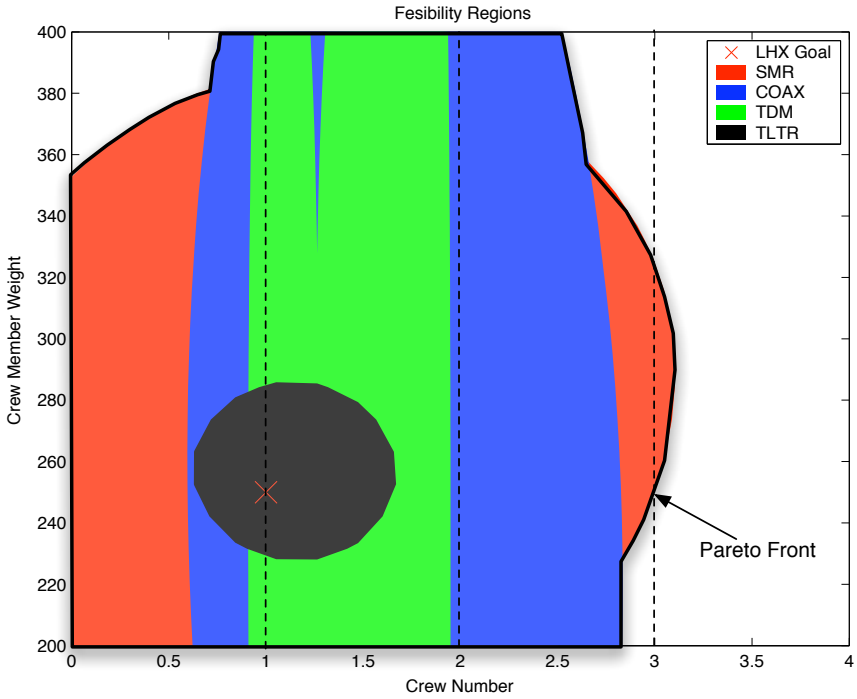


Figure 55: Original LHX Requirements Pareto Front

would be forced to switch to either the SMR or COAX systems. If the number of crew was then decreased and the crew weight increase d, shown in Figure 54(d), at no point would a switch back to the TDM system be required.

This is a good example of hysteresis in the discontinuous system type boundaries mentioned in Chapter 1. It should be noted that this fully rigid "jumping" assumes that there are no preferences for any of the vehicle types, or any specific settings of the state variables that will change as the number of crew or the crew member weight increases or decreases. If preferences are added, then some of the rigidity will be lost, changing the points where the jumps between system types occur, moving them away from the technology driven boundaries.

It is also possible to superimpose the requirements Pareto front upon this chart. This is shown in Figure 55. The Pareto frontier describes the combinations of state-vectors, and thereby systems that proscribe the best combinations of requirements

Table 16: Mission Segment Alterations, 1983 LHX vs. RAH-66 Comanche

Mission Segment	LHX Time	RAH-66 Time
2	15	25
3	20	25
5	15	25
7	20	25
16	20	25
17	20	25
19	30	35

that the technology allows. Typically, the Pareto front is defined only for the "good-good" of front-side of the space. However, since no preference of direction of improvement is implied in the MSPEA tool, it is appropriate to label all sides of the feasible region as belonging to the Pareto frontier.

6.3.4 Current RAH-66 Comanche Technology Limits

Over the course of the LHX/RAH-66 Comanche program, not only did the system level requirements change, but also the technology level of the vehicle's subsystems and components. The primary differences between the original LHX vehicle and the resulting RAH-66 Comanche can be seen by comparing Table 9 with Tables 1 through 8. In addition to the increase in gross-weight and armament that occurred over the life of the program, the basic range also increased from approximately 170 nautical miles to approximately 260 nautical miles [116, 120]. Because of the implementation of this example it was necessary to change the time lengths of several of the mission segments to accommodate this increase in range. The new mission segment times are given in Table 16. Further, the utility vehicle was dropped from the LHX requirements during the course of the program; therefore, it was not necessary to investigate the ability of the Comanche to meet the utility mission. Therefore, the corresponding utility mission segment times, segments 8-15, were zeroed. Since the improvements in the technology limits shown in Table 12 were primarily the result of changes in the state-of-the-art that were independent of the LHX program it is of interest whether

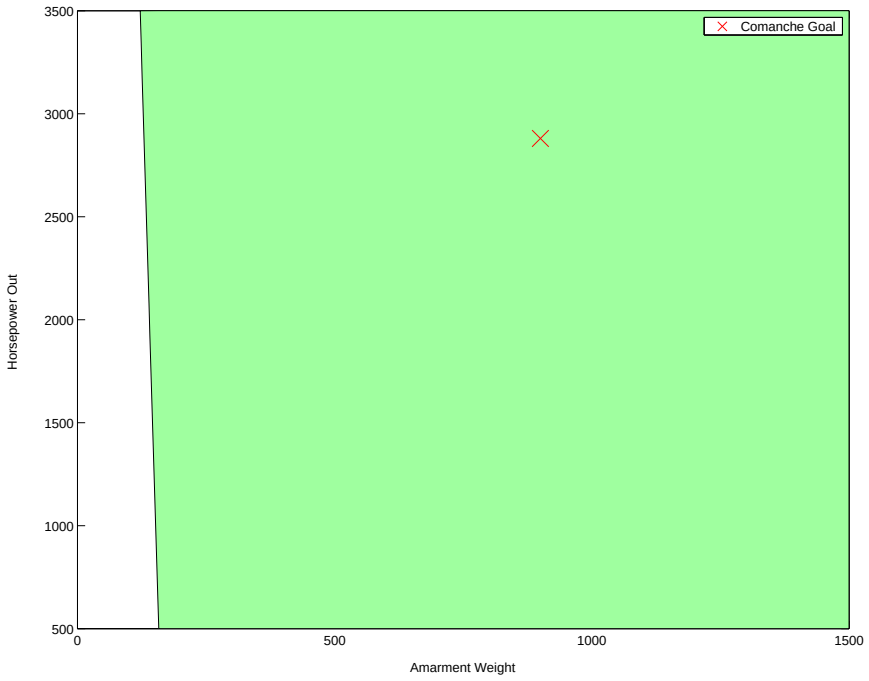


Figure 56: SMR Feasible Regions, RAH-66 Requirements, 1983 Technology Limits, Armament Weight vs. Horsepower

or not the original technology limits would have enabled the RAH-66.

6.3.4.1 Original Technology Limits

The original 1983 technology boundaries produced a relatively large feasible region at the 1983 LHX requirements values. This can be seen from the SMR vehicle results shown in Figures 49 through 51. For this reason, it was decided to investigate whether or not the original technology limits allowed for the requirements to which the RAH-66 was ultimately designed. The results were surprising, considering that the technology level of the RAH-66 has advanced past what the original 1983 vehicles were based on. The results for the feasible regions in the requirements (hyper)space using the 1983 technologies and RAH-66 requirements are shown in Figures 56, 57, and 58. Since the size and shape of the feasible regions has changed with the new mission and load-out requirements it is beneficial to see how the sensitivity of the feasible region behaves. This is shown in Figure 59.

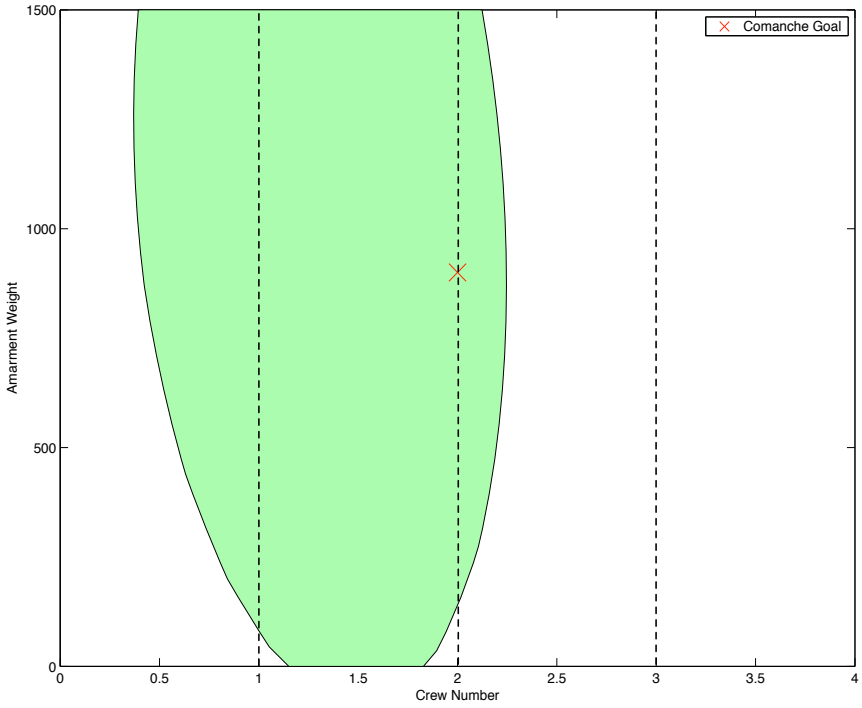


Figure 57: SMR Feasible Regions, RAH-66 Requirements, 1983 Technology Limits, Number of Crew vs. Armament Weight

Since the original technology limits allow the SMR vehicle type to meet the RAH-66 Comanche's requirements, it is of interest why the technology levels were adjusted. Was it because the available technology improved enough to make the system more efficient, and allow future growth, or were there other requirements that existed which cannot be analyzed by the current $\mathbf{R_f}$ tool?

6.3.4.2 Current Technology, SMR Vehicle System

The RAH-66 is an advanced technology, SMR system. Therefore, it is useful to demonstrate the technical feasibility of the design in the requirements (hyper)space. Obviously, the RAH-66 as it is designed meets the requirements it was to be procured under. If the vehicle had been unable to meet the requirements either the vehicle and/or the requirements would have been adjusted to bring the two into agreement. Consequently, another validation point is obtained if the results from the MSPEA

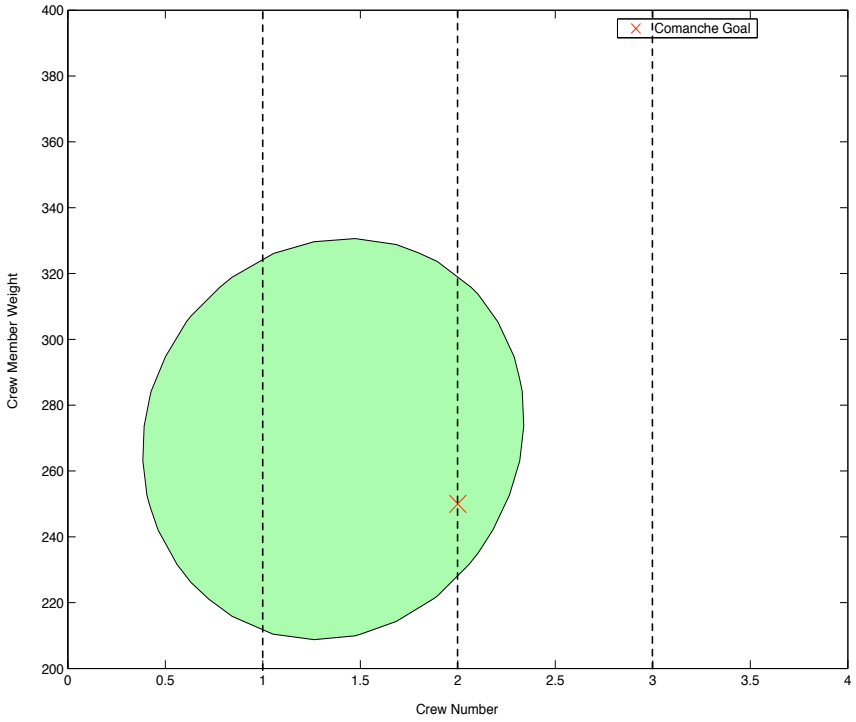


Figure 58: SMR Feasible Regions, RAH-66 Requirements, 1983 Technology Limits, Number of Crew vs. Crew Member Weight

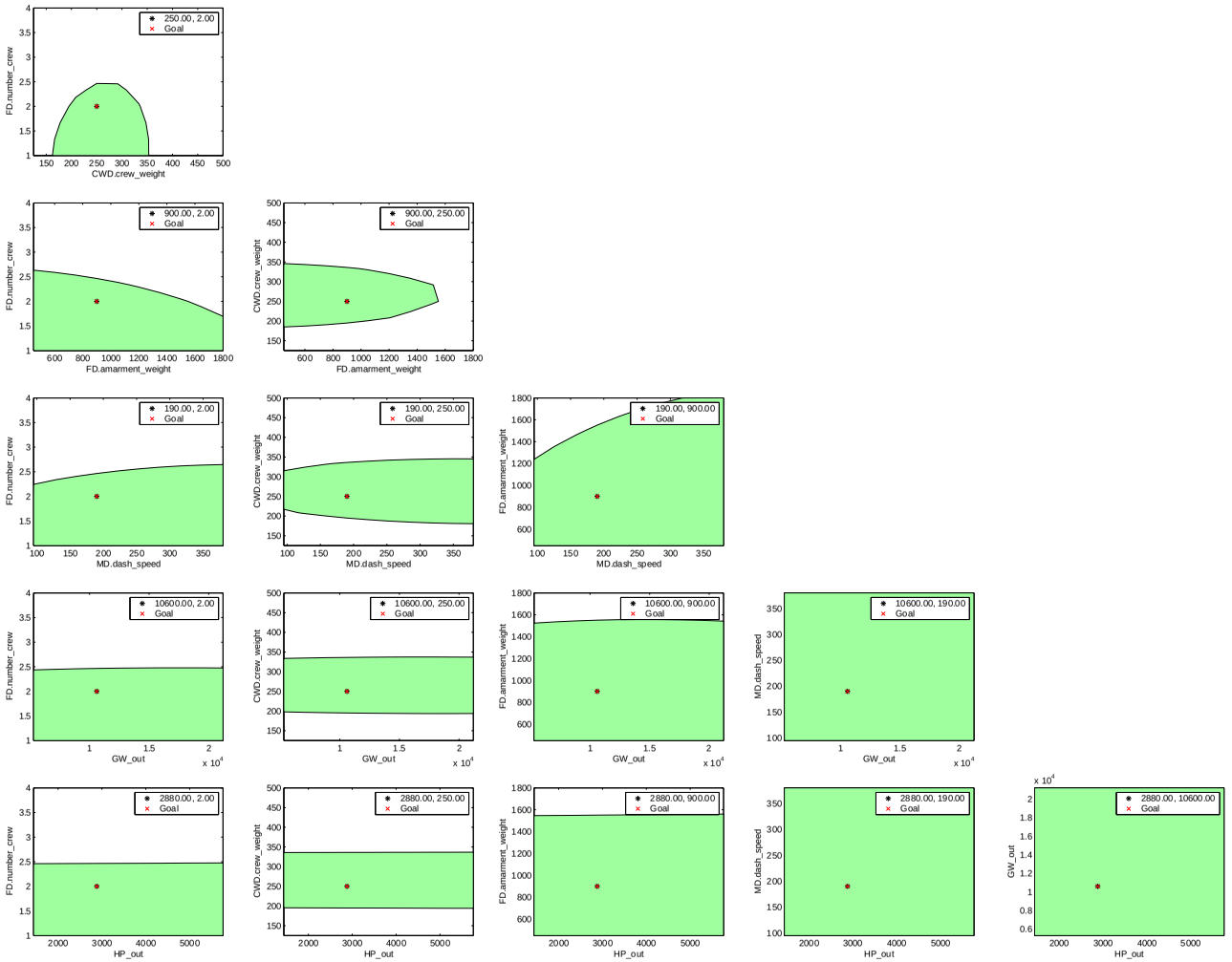


Figure 59: SMR Feasible Region Sensitivity, RAH-66 Requirements, 1983 Technology Limits

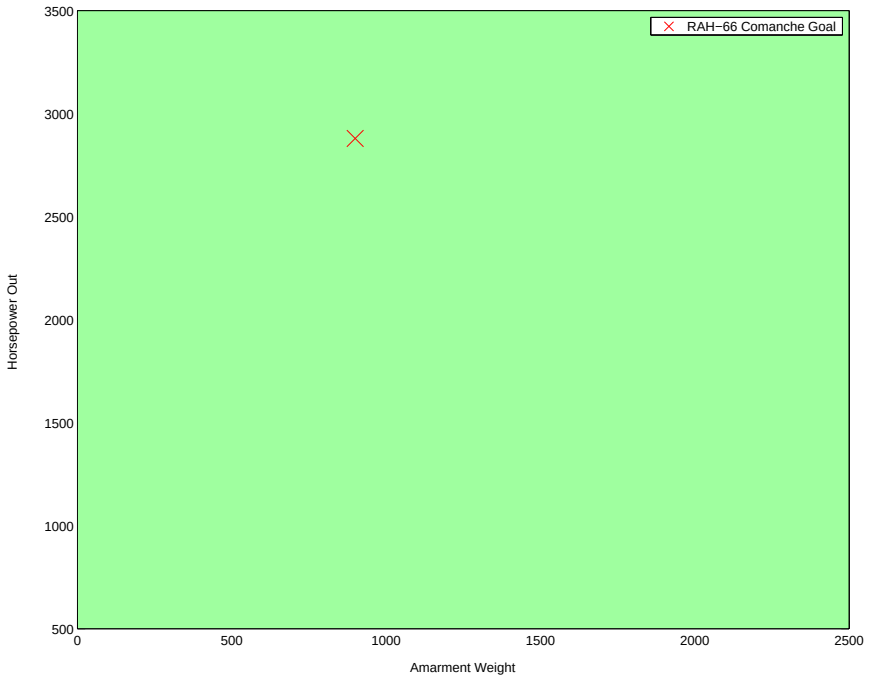


Figure 60: SMR Vehicle, Current Technologies, RAH-66 Requirements, Armament Weight vs Horsepower

tool demonstrates the feasibility of the system using the current technology limits.

One of the benefits of investigating the Comanche is that the transport/utility version of the LHX was dropped during the course of the program. Therefore, it was possible to "zero out" those segments of the composite mission for the RAH-66 investigation. This effectively reduces the number of requirements dimensions investigated from 43 to 32. This significantly reduces the size of the space; and therefore, the complexity of the visualization models. The resulting plots of the feasible requirements region, corresponding to those in Figures 48, 49, and 50, are shown in Figures 60, 61, and 62 respectively. It should be noted that the minor change in technology, specifically relating to the power-plant significantly opens the available requirements space for the RAH-66 Comanche. By looking at the sensitivity of the feasible space to different requirements this becomes even more obvious. Compare the results shown in Figure 51 to those shown in Figure 63. The effect of changing the requirements

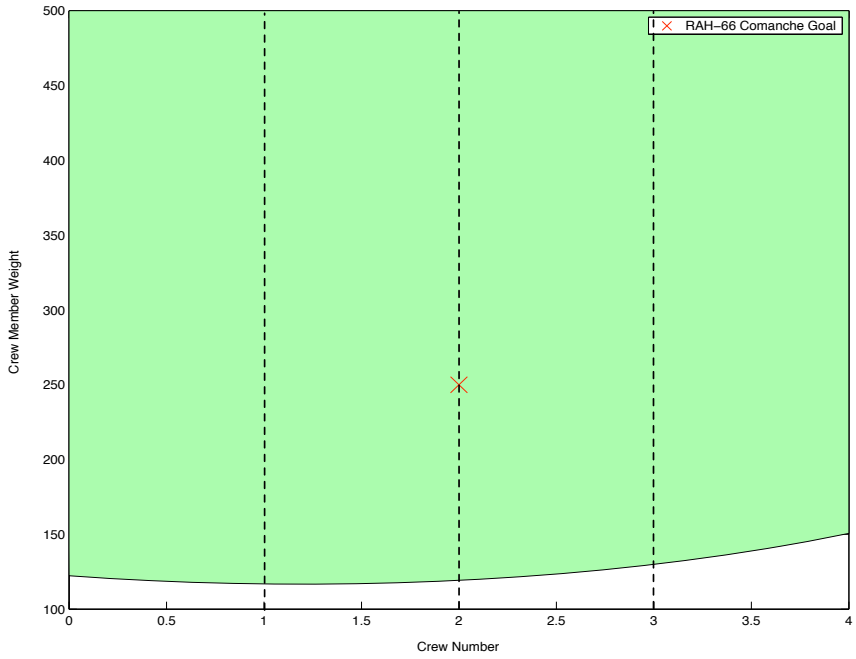


Figure 61: SMR Vehicle, Current Technologies, RAH-66 Requirements, Number of Crew vs Crew Member Weight

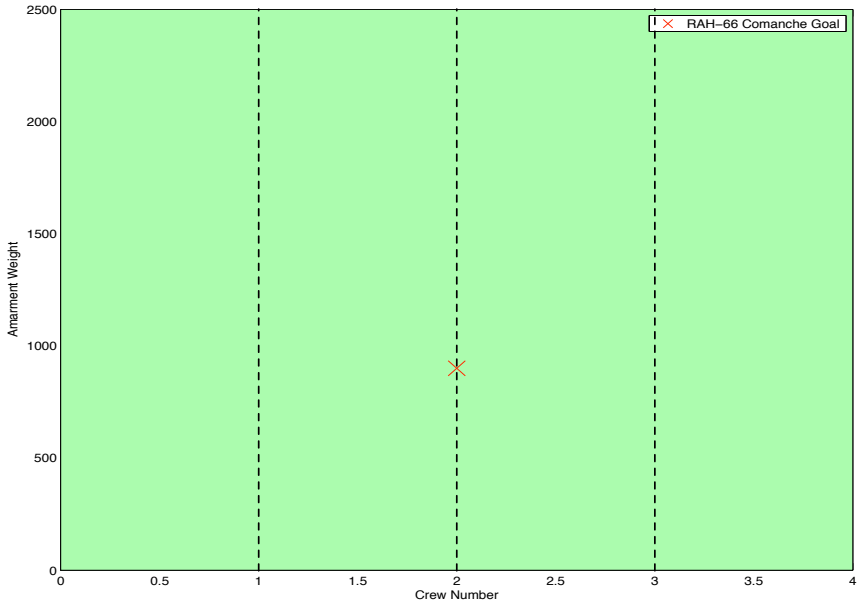


Figure 62: SMR Vehicle, Current Technologies, RAH-66 Requirements, Number of Crew vs Armament Weight

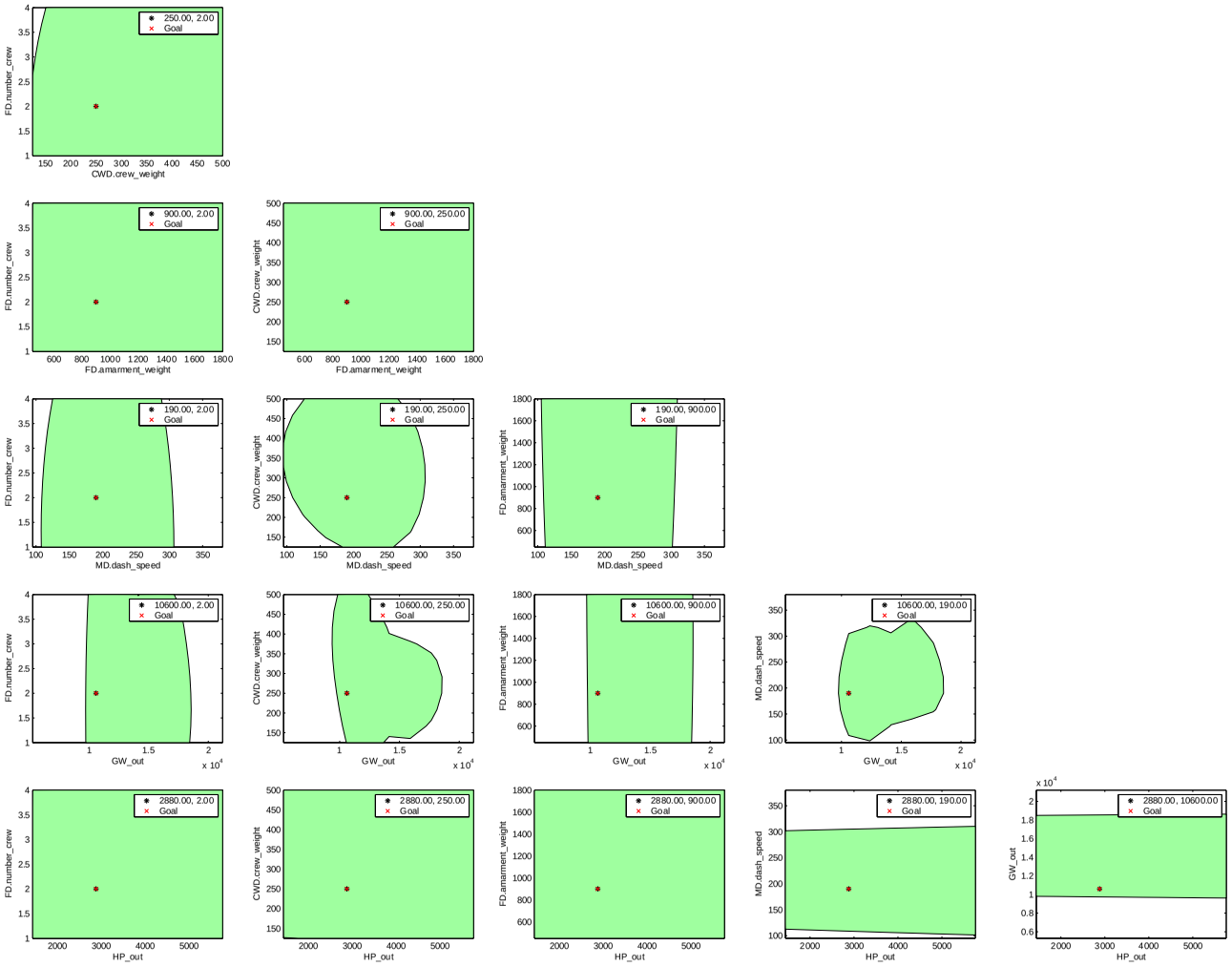


Figure 63: SMR Requirements Sensitivity, Current Technologies, RAH-66 Requirements

and technology levels is easily seen by looking at the dash speed vs. number of crew members and crew member weight charts.

Another aspect of the revised requirements (hyper)space is that the RAH-66 requirements no longer exist near the boundary of the feasible space, but are now located more toward the interior of the feasible region. There are several reasons that this may happen, particularly the fact that another requirement or technology boundary, i.e. one not considered is actually driving the location of the vehicle in the feasible region. Since the data available for the RAH-66 Comanche is less than that available for the original LHX, this could easily be the case. Furthermore, since the RAH-66 is a fully designed and prototyped vehicle, the fidelity of the analysis that has driven the vehicle to this location in the requirements (hyper)space is significantly higher than that of the $\mathbf{R_f}$ tool. Changing the underlying tools effectively changes the behavior of the system, and therefore, may change the location of the technology boundaries in the requirements (hyper)space.

6.3.4.3 Additional Vehicle Types

The results for the remaining single vehicle type explorations have been placed in Appendix G as they do not, individually, provide any significant new understanding about the problem.

6.3.4.4 Current Technology, Multiple Vehicle Systems

The multiple vehicle system superposition in a slice of the requirement (hyper)space for the RAH-66 is similar to that shown for the original LHX requirements in Figures 52 and 53. The same multiple system projection on a slice of the requirements (hyper)space is shown in Figure 64. Again since the feasible spaces overlap, it is necessary to see the location of the hidden boundaries. These are shown as lines in Figure 65.

Again, no direct elimination of a system type is possible from the results shown

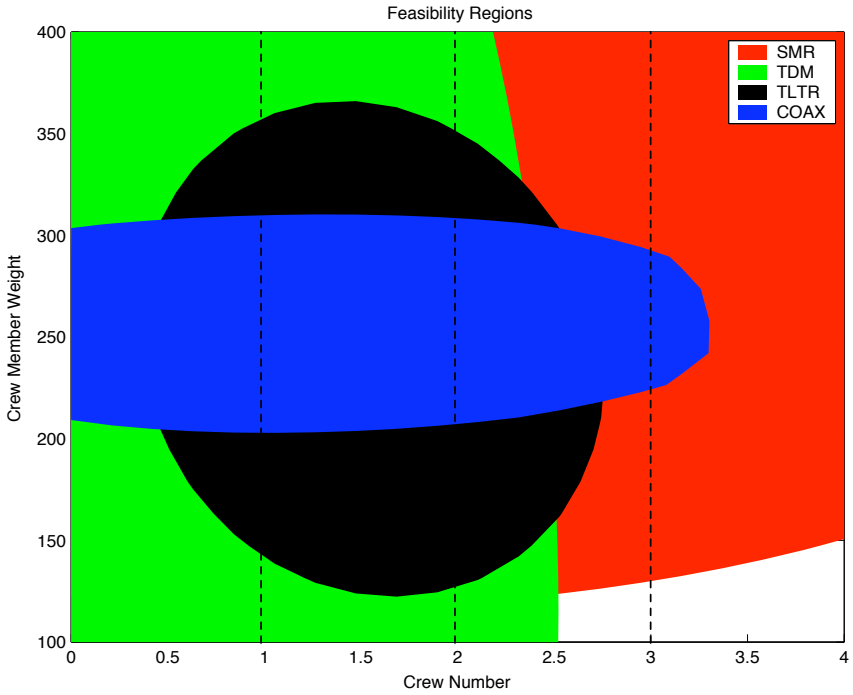


Figure 64: TDM Vehicle, Current Technologies, RAH-66 Requirements, Number of Crew vs Crew Member Weight

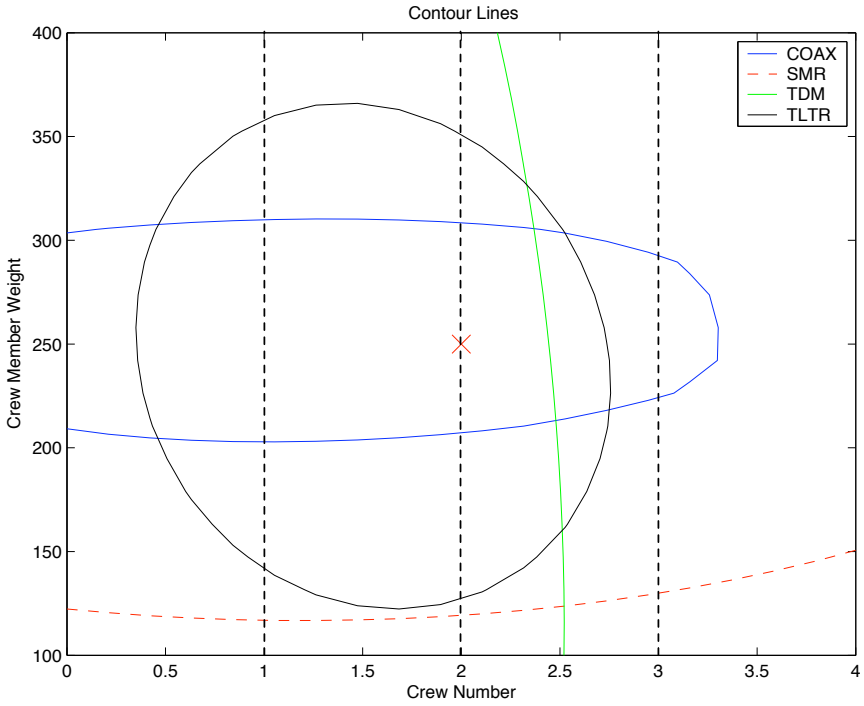


Figure 65: Multiple Vehicle Type Feasible Requirements Boundary Contours, RAH-66 Requirements, Number of Crew vs Armament Weight

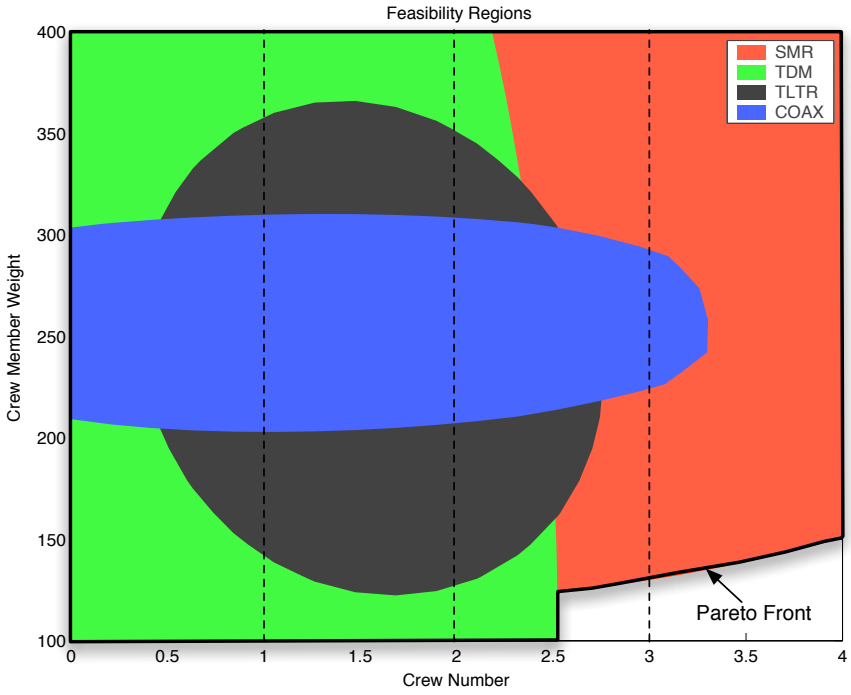


Figure 66: Current RAH-66 Requirements Pareto Front

in Figure 64 as the RAH-66 requirements point lies within the feasible space for all four vehicle types. In reality all but the SMR vehicle type were eliminated. As to why this happened the author can only speculate. It may have been driven by other requirements or preferences or the capability of a specific technology that didn't live up to its billing. These are all sources of programmatic uncertainty. Even with this result it is still possible to play similar "what if" games with the RAH-66's requirements.

Again the requirements Pareto front can be determined from these figures. For the RAH-66 this is shown in Figure 66

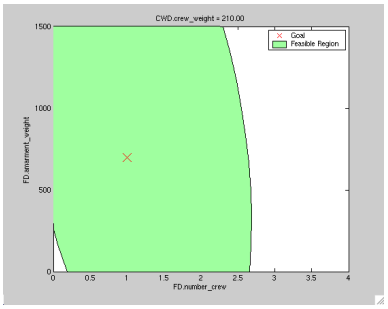
6.3.5 Further Visualization Possibilities

Static two-dimensional representations, while appropriate for a written document, fail to unleash the full capabilities of the boundary discovery and visualization techniques. To do this, it is necessary to investigate the response of the planar feasibility space as

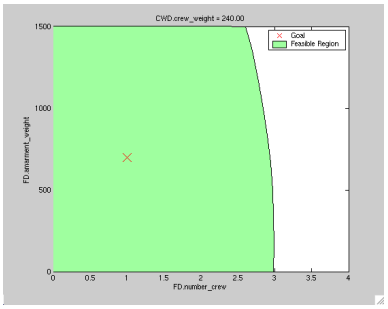
other variables are changing, i.e., effectively viewing three, four, or more dimensions simultaneously. If the visualization meta-model is simple enough, and the resolution desired is low enough, then this can be performed in real time. However, for the LHX meta-models this is not the case. It takes several seconds to produce a relatively low resolution contour chart for a single vehicle type. A solution to this temporal problem presents itself. If the range of interest is known ahead of time a movie of the different slices can be made, and scrubbed through to give the decision maker a feel for the effect that changing one or more of the "background" requirements has to the feasible region. A movie of this sort can be created for one or more vehicle types, with one or more "background" requirements.

6.3.5.1 Single Vehicle Type, Single Background Requirement

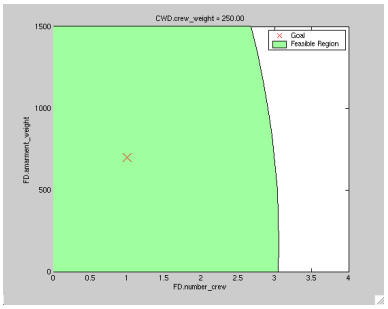
This simplest example of the movie is for a single vehicle type with a single "background" requirement varying. In this case the basic SMR vehicle type, in the original 1983 LHX requirements (hyper)space was chosen. The requirements that are displayed are the number of crew and the armament weight. The effect of changes in the weight of each crew member on the feasible region was then investigated. Since it is not practical to present a movie in a fixed paper format several "pictures" were taken at different settings of the "background" requirement. These are shown in Figure 67 and display the feasible region on a slice comparing the number of crew members vs. the armament weight. The feasible region shown in Figure 67(a) encompasses most of the left portion of the chart. Only a small region of infeasibility exists in the lower left-hand corner. This infeasible region disappears as the crew member weight is increased, shown in Figures 67(b), 67(c), and 67(d). However, the infeasible region returns for a crew member weight of 340 lbs, shown in Figure 67(e). Finally as the crew member weight grows further the feasible region begins to shrink drastically, as exhibited in Figure 67(f). The interesting thing about this progress is that both



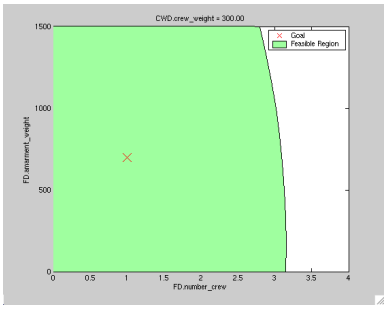
(a) Crew Weight = 210 lbs.



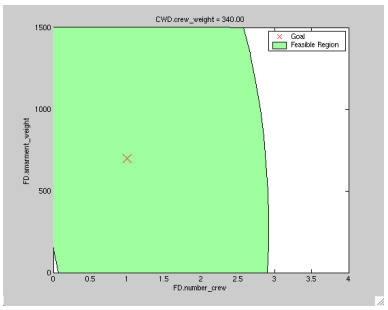
(b) Crew Weight = 240 lbs.



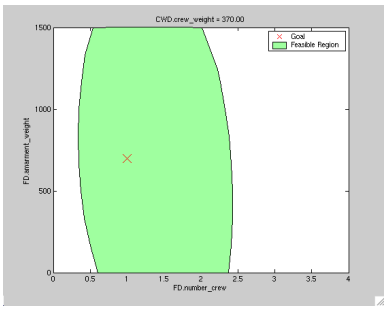
(c) Crew Weight = 250 lbs.



(d) Crew Weight = 300 lbs.



(e) Crew Weight = 340 lbs.



(f) Crew Weight = 370 lbs.

Figure 67: Single Vehicle Type, Single Background Requirement, Effect on the Feasible Region

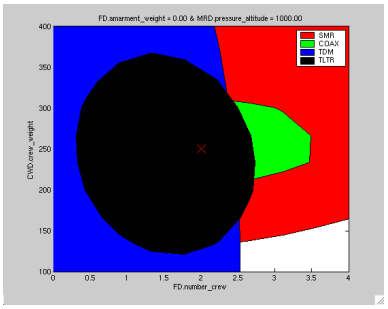
the "frontside" and "backside", explained earlier in this chapter, of the crew-member weight requirement is visible. This was described previously and may or may not affect the choices made.

6.3.5.2 Multiple Vehicle Types, Multiple Background Requirements

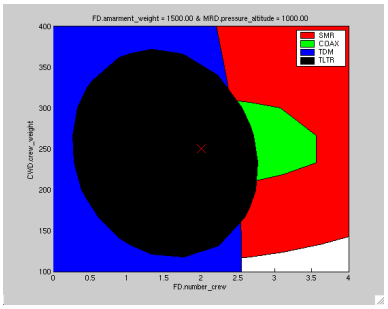
Increasing the number of vehicle types being investigated and the number of "background" requirements being varied increases the complexity of the visualization; however, it also provides significantly more information. Because of the fact that the RAH-66 requirements coupled with the current technology boundaries produced feasible regions for all four vehicle types it was decided to perform the multiple vehicle type, multiple background requirements demonstration using this requirements set and vehicle type combination. The results are shown in Figure 68. The most obvious result is that all of the vehicle types remain feasible throughout the entire investigation. Of course only two of the 41 total background requirement input variables are being varied; therefore, no sensitivity to any of the other background variables can be implied. This means that the feasible space for any vehicle type might shrink rapidly if any of the other 39 requirements are varied.

6.3.5.3 Using MSPEA to Determine Potential State Vectors

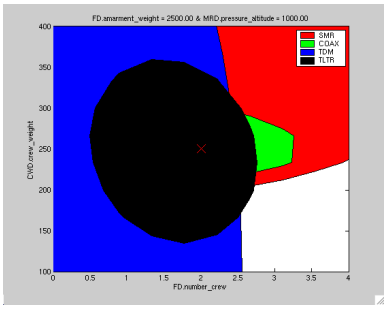
In addition to producing movies on the effects of changing background requirements, it is possible to obtain other information from MSPEA. The method for determining the technology limit driven system boundaries with the MSPEA tool is based upon discovering the frontier, created by the technology boundaries in the state/design space, in the requirements (hyper)space. Since the MSPEA assumes no preference for the different requirements, there is no guarantee that specific states will be found for any point along the boundary. Further, in many cases additional requirements, either not known or envisioned, or not modeled by the underlying tool, may constrain the choice to a region inside the boundaries. In these cases the decision to only find the



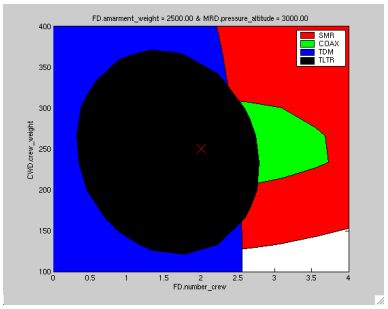
(a) Armament Weight = 0 lbs, Rotor Sizing Altitude = 1000 ft



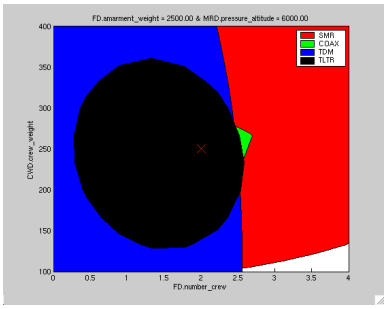
(b) Armament Weight = 1500 lbs, Rotor Sizing Altitude = 1000 ft



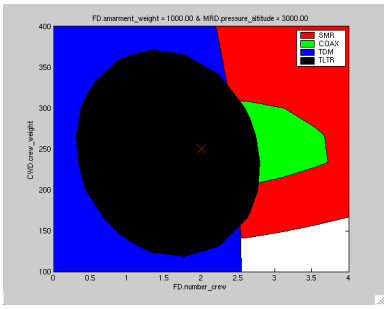
(c) Armament Weight = 2500 lbs, Rotor Sizing Altitude = 1000 ft



(d) Armament Weight = 2500 lbs, Rotor Sizing Altitude = 3000 ft



(e) Armament Weight = 2500 lbs, Rotor Sizing Altitude = 6000 ft



(f) Armament Weight = 1000 lbs, Rotor Sizing Altitude = 3000 ft

Figure 68: Multiple Vehicle Types, Two Background Requirements, Effect on the Feasible Region

system boundary means that no knowledge of the potential states is readily available.

The MSPEA tool is inherently capable of determining the range of states available at a specific setting of requirements. Since the MSPEA tool is designed to find a front in one space, based upon a front in another space, the tool has no inherent limitation as to which mapping it will seek. Therefore, to determine the boundary of available states all the user has to do is swap the inputs for the outputs, i.e. the specific requirements setting would now be the goal, and the state variables would be the input parameters in the MSPEA tool. This would produce a broad, nonclustered frontier in the state space, using the same equilibrium principles that are used in RCD.

A simpler and less time consuming option, for determining the available states at given requirements points is to cull the outputs from the MSPEA tool when it is used for the technology driven system boundary discovery. Since the MSPEA tool is designed to keep unique solutions in both the requirements and state variables, there is a significant chance that multiple solution vectors for a single requirement setting will be present in the output file. If these match or are close to the settings desired, the engineer or decision maker can use this information with no additional computational cost. An example of this is given in Figure 69. The results for this figure were culled from the 1983 LHX, SMR vehicle type boundary discovery.

Looking at Figure 69 it is evident that all three of the vectors are very similar, differing at only two points. As an example, look at the two setting for *MRD.blade_number*, which corresponds to the number of blades in the main rotor. The two settings which are present correspond to -0.35 and -0.55 of the normalized range. Comparing this with the range given in Table 14, on page 109, these represent a setting for the number of main rotor blades of 5 and 4 respectively. While either combination can be taken as valid, no information as to the feasibility of other numbers of main rotor blades can be implied. In order to obtain this information a more in-depth look at

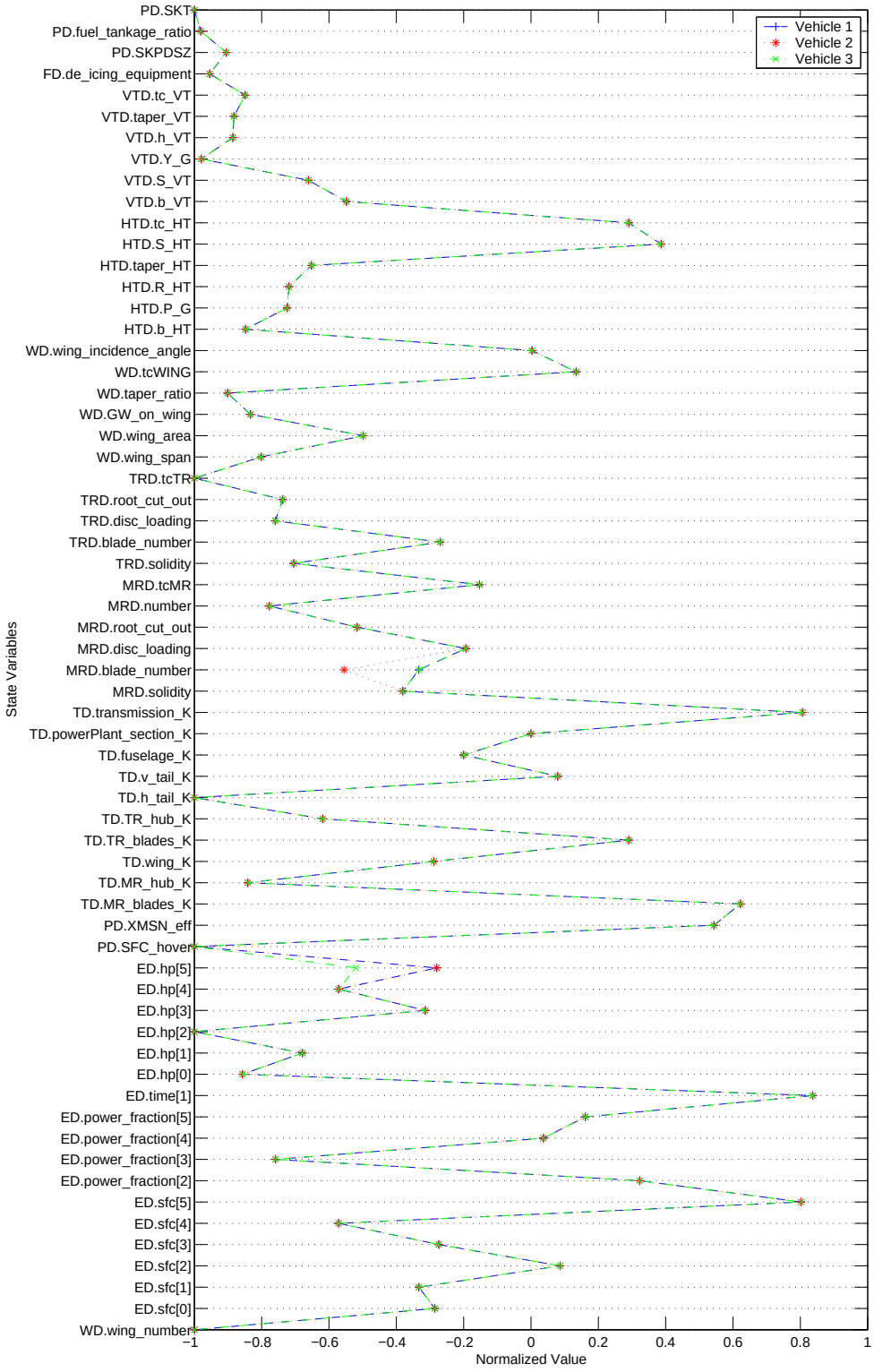


Figure 69: Example State Vector, Single Requirements Setting, Culled from MSPEA Output, 1983 SMR Vehicle Type

this single requirements point would have to be performed.

6.3.6 Combinatorial Feasibility

The benefit of using the boundary discovery methodology implemented in the MSPEA tool is that it significantly reduces the computational complexity of the problem when compared to a brute force, grid search approach. Comparing the results shown in Appendices B and C, it is quickly possible to see the effect of increasing dimensionality and resolution in the requirements (hyper)space. This, however, does not tell the entire story. Since RCD requires an inherently "equilibrium" methodology, any approach must take into consideration that multiple solution vectors may exist at any point in the requirements (hyper)space. If the limits of the system type and model are not taken into account, there may be hundreds of potentially feasible system states at any given point in the requirements space. This is inherently different from the "frozen" methods which will generally remain within their "well" in the space, meaning only one solution vector will present itself for a given perturbation in the requirements. Consequently, any investigation of the requirements (hyper)space needs to consider both the possible settings for the requirements/control and the design/state vectors.

6.3.6.1 Comparison of Computation Times for LHX Example

The input vectors to the $\mathbf{R_f}$ tool for the original 1983 LHX program consisted of 43 requirements and 64 state variables. Each of the variables was discretized for inclusion in the MSPEA tool. The number of bits for each variable are presented in Tables 10, 11, 13, and 14. The total number of bits in the resulting "chromosomal bit string" is 144 for the requirements and 250 for the state variables. Combining these gives a total bit string length of 394. Considering the $\mathbf{R_f}$ tool is capable of evaluating approximately 40 unique state vectors per CPU second, the time it would take to fully evaluate all possible combinations, presented in Table 17, is astronomical. Even

Table 17: LHX Computation Time Comparisons, Single Vehicle Type

Method	Bits	Cases	Seconds	Years
Grid Search	394	4.03×10^{118}	1.01×10^{117}	3.2×10^{109}
MSPEA	394	~ 5125	~ 7200	-

Note: The runtime of the MSPEA algorithm is heavily dependent on the number of internal population members and the number of generations. The numbers represent 250 internal members and 75 generations. Case scaling is less than the number of generations×population as the strengths are stored.

reducing the number of points investigated in each of the 106 dimensions to four entails calculating 6.6×10^{63} points. This is still entirely impractical.

It is only by eliminating the desire to possess information for the entirety of the requirements (hyper)space that the problem becomes at all feasible. Still the points investigated by the MSPEA routine are an infinitesimal portion of the entire space. There is, therefore, the possibility that a feasible point is going to be missed. However, considering the increase in capability over the existing methods this potential drawback is most likely worth it. Further, the risk of missing an important region decreases as the number of internal and external points in the MSPEA tool is increased. Therefore, a balance between runtime and uncertainty needs to be struck. This balance is left to the engineer or decision maker.

6.3.6.2 Capability of Investigating the Requirements Hyperspace Using a Grid Search

It is of interest whether or not the MSPEA results match those determined through a grid search. Appendices B and C demonstrate the capabilities of both the grid search method and the simple GA; however, because of the increased complexity of the MSPEA tool it would be beneficial to demonstrate the location of some feasible states in a reduced requirements space. This validation was performed in two steps. First the number of requirements that were varied was reduced to two, and several thousand cases were run across this grid. Second the variability of the requirements was reduced to zero dimensions, and several hundred cases run at the original LHX

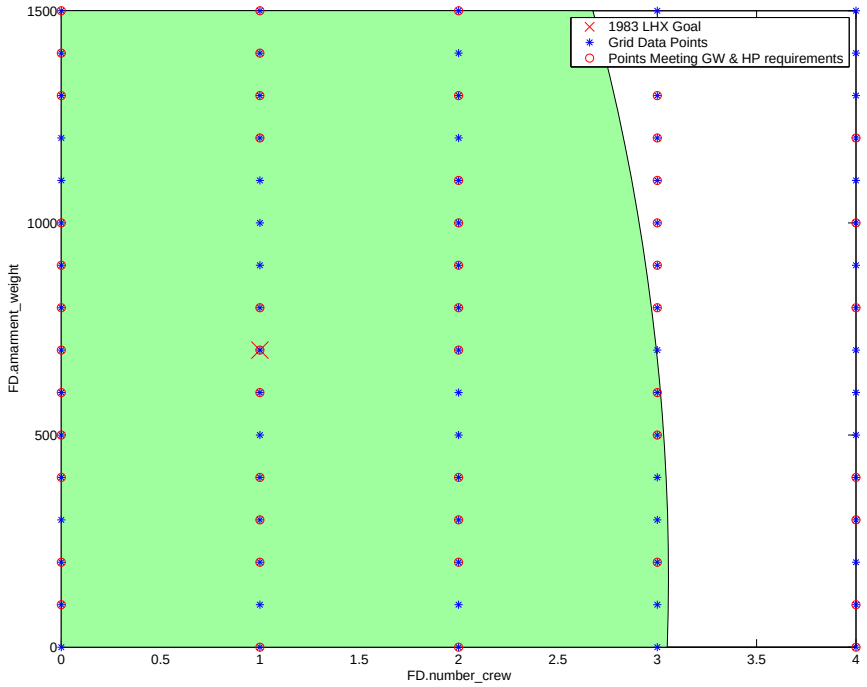


Figure 70: Results of the Grid Search for Two Requirements, 4096 Cases

point.

By reducing the number of requirements that were allowed to vary to two, the total combinatorial complexity of the requirements space was reduced by approximately 52 orders of magnitude. The requirements chosen were crew number and armament weight, as shown in Figure 50 on page 119. The space was then gridded with eight points on the number of crew axis, $0-7$ and 16 points on the armament weight, $0-1500$ lbs. This created a total of 128 grid points in the requirements space. Since the process is inherently an "equilibrium" one, and the number of varying state/design variables was 63, it was decided to initially investigate 4096 separate points in the requirements and design spaces. The results from this investigation were reduced to ensure both that the points met the requirements that were $\mathbf{R_f}$ tool outputs, and within the 1983 technology boundaries. Of the original 4096 points investigated none met all of the criteria. A graphical presentation of this is shown in Figure 70. The green area in the figure represents the feasible requirements space as discovered

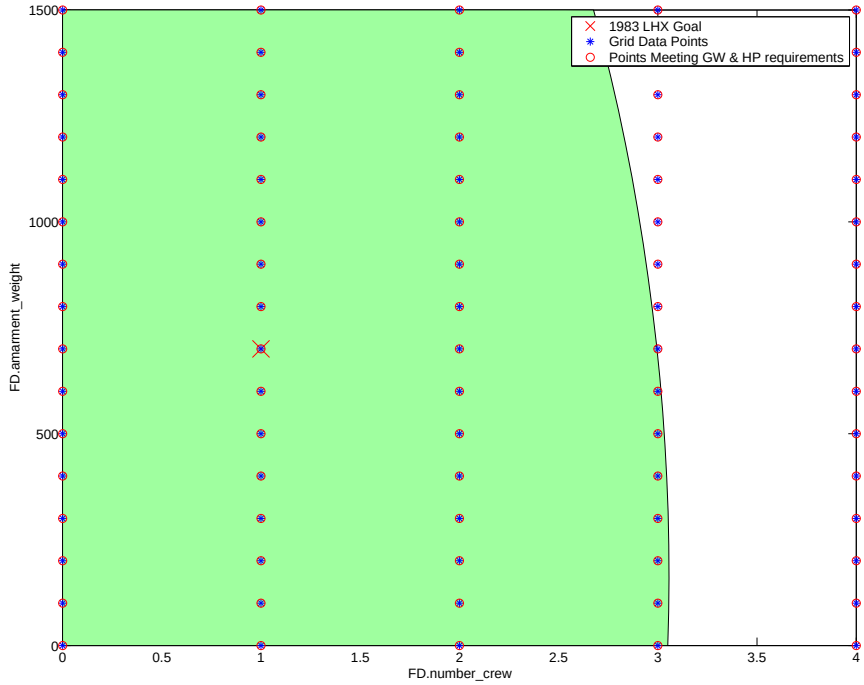


Figure 71: Results of the Grid Search for Two Requirements, 8192 Cases

by the MSPEA algorithm. The runtime of this unsuccessful "brute-force" approach was approximately 1/3 that of a single run of the MSPEA algorithm for the LHX validation problem.

Since the grid search was unable to find a single feasible point out of the original 4096 cases is was decided to increase the number of cases to 8192 in hopes of increasing the chance of finding a feasible point. Again of the 8192 cases, zero proved capable of meeting both the $\mathbf{R_f}$ output requirements and the technology boundaries. These are presented in Figure 71. Why was this "brute-force" approach unsuccessful? There are approximately 3.6×10^{103} possible state vector settings at each of the 128 requirements points and the number of feasible states at each of these point is likely a small fraction of the total. Therefore, there is an extremely low likelihood that a point investigated is one of the feasible subset.

The next "sanity check" was to investigate a multitude of state vectors at the original 1983 LHX requirements point. This was done in hopes of obtaining some

knowledge about the percentage of the 3.6×10^{103} possible vectors that lie within the feasible region. This time only 1024 cases were run. Of these 1024 cases only 83 fulfilled the two requirements that were $\mathbf{R_f}$ tool outputs, and none were fully within the technology boundaries. Again this is not surprising considering what a small portion of the state space these 1024 cases represent. Nonetheless it is informative to view the different values of the state variables that were taken for the 83 remaining cases. Normalized values of these and the technology limits are shown in Figure 72. The green dots represent state variable settings present within the data set. Additionally, three example vectors are plotted. These represent a few of the possible combinations. For many of the state variables the majority of points lie outside the technology boundaries. This seems to indicate that only a tiny portion of the total state space will fulfill the LHX requirements.

It is only by significantly reducing the amount of state variables, and their potential values that it is possible to produce a grid search which provides some feasible results for the LHX program. The number of state variables that were allowed to equilibrate was reduced to 12, reducing the total bit string length to 41. This results in approximately 2.2 trillion potential combinations at each of the 128 grid points. Another 4096 cases were run and these were culled to find the points that met both the output requirements and the technology boundaries. The remaining points, 197 in all, are shown in Figure 73. Some of the grid points have more than one feasible case. All of the points that were produced in the reduced dimensionality grid search and that met the technology limits are within the region of feasibility identified by the MSPEA algorithm. Further, the algorithm was able to identify a larger portion of the requirements region than even the reduced complexity example. This truly indicates that except for cases with a very small number of requirements and state variables the grid search is totally impractical. The expansion of a single requirement point, shown in Figure 72 was repeated, and the results are shown in Figure 74. Again none

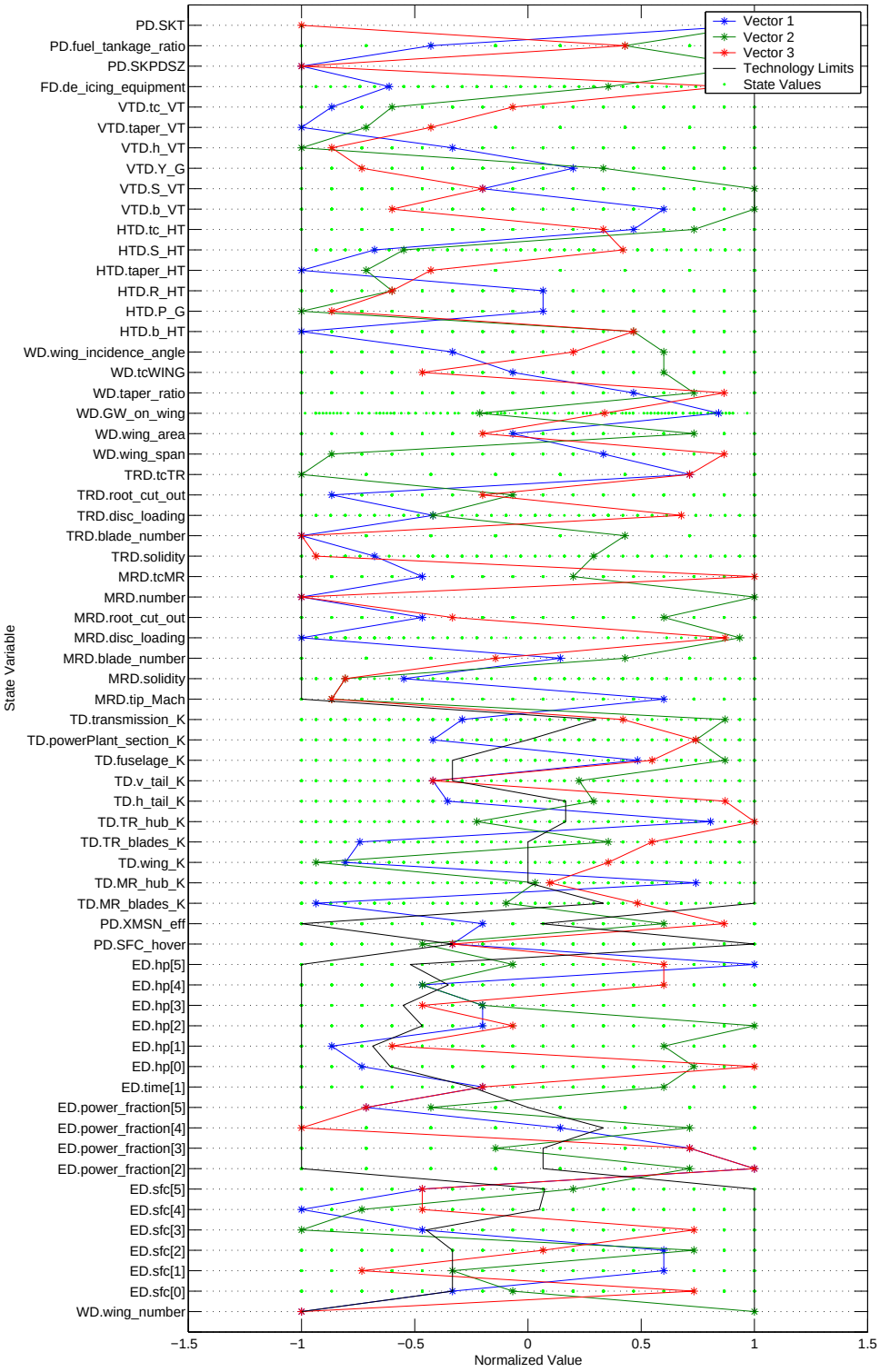


Figure 72: Grid Search Results, Single Requirements Vector, State Variable Settings, SMR Vehicle Type

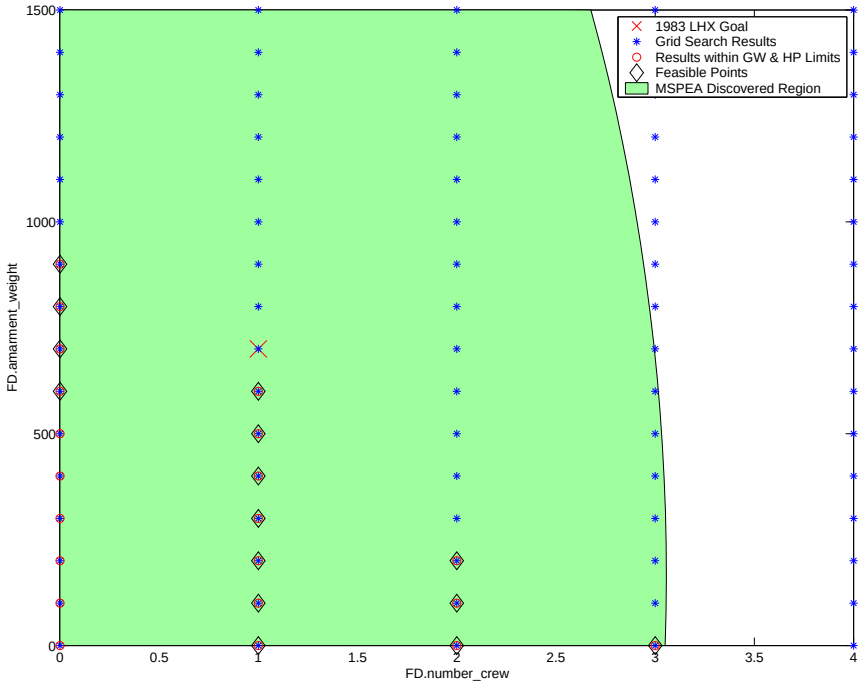


Figure 73: Results of the Grid Search for Two Requirements, 12 State Variables, 4096 Cases

of the original 1024 values, met both the output requirements, GW & HP, and the technology limits. This is not worrisome as the original SMR vehicle type sizing point listed by Keys was initially verified [116, pp. 98, 99, 123]. This only further shows that if the number of cases investigated is only a small fraction of all of the potential states there is good chance the grid search will miss the feasible points.

6.3.6.3 Computational Limitations of RCD and the MSPEA Tool

The MSPEA tool does not suffer from the same "curse of dimensionality" that the "brute force" grid search method does. This arises from the fact that the technology limit induced boundary takes up a smaller and smaller fraction of the total requirements (hyper)space as the dimensionality of this (hyper)space increases. This fact should not be taken to imply that there is no practical limit to the number of requirements that can be investigated simultaneously. Because the number of points that

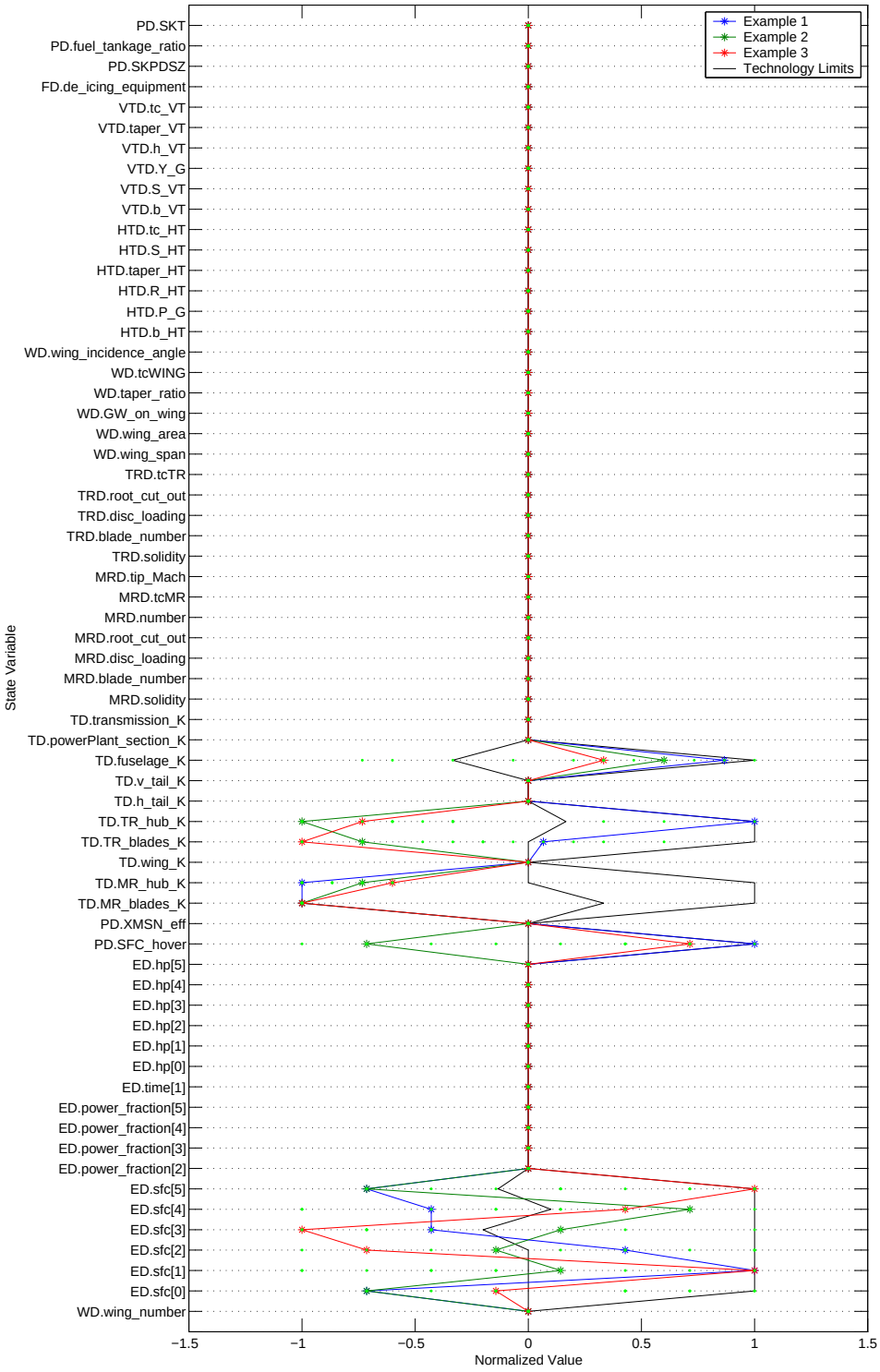


Figure 74: Grid Search Results, Single Requirements Vector, State Variable Settings, 12 State Variables, SMR Vehicle Type

the MSPEA tool investigates is determined independently from the number of dimensions, if the internal population remains fixed while the bit string length increases, there will be a greater risk that a useful feasible point will be missed. Further, as the dimensionality increases there is a corresponding need to increase the number of points included in the visualization meta-model. This was demonstrated with the difficulty in fitting the meta-model for the LHX validation example. These difficulties were not encountered during testing on notional systems with few requirements.

The combination of a need for more internal and external MSPEA points with the runtime scaling of both the MSPEA tool and the visualization tools imposes a practical limit on the number of requirements that can be satisfactorily investigated. Because the runtime is so dependent on the underlying tool, in this case the $\mathbf{R_f}$ tool, it is up to the engineer or decision maker to determine the acceptable combination of internal and external points, with MSPEA runtime, visualization runtime and accuracy. Because of the inability to properly investigate the loss of fidelity in the meta-models as dimensionality increases, the grid search validation scheme quickly becomes combinatorially impossible and the standard "frozen" methods are completely incapable of providing knowledge, and no specific warnings or tips can be issued.

6.4 Final Comments on the Validation Cases

The validation cases presented in this chapter helped identify and confirm the strengths and weaknesses of using an EA in RCD. The primary limitation with the EA approach is that there is a risk, which depending on the dimensionality may be significant, that a worthwhile portion of the feasible space may be missed. The risk of this occurring is dependent upon both the number of internal and external population members. Since the internal population in the MSPEA is the working population, the smaller it is with respect to the total number of possible points the greater the risk that a potential solution will be missed. This is a problem with any EA/GA optimization

method and is well recognized. Conversely, the external population controls the accuracy of the visualization model. Since the meta-model is only as good as the data included in it, the fewer the number of data points included in the visualization model the higher the uncertainty of its predictions. One of the benefits of using a Gaussian process is that it has a relatively robust error estimation capability. The downside to including more and more points as the dimensionality of the problem increases is that the visualization runtime increases significantly. Therefore, a trade-off between the risk of missing useful regions and runtime needs to be made.

While the risk of missing a feasible area when using the MSPEA method is of concern, it must be offset against the greatly improved capability that is derived from using an EA over the grid search technique, or the current methods. The current reliance on "frozen" methods completely eliminates the ability to investigate a broad scope of requirements, systems, and states simultaneously. Consequently, RCD opens up a significant capability with an associated set of caveats. As long as the number of requirements is kept to reasonable number, most likely in the 10-25 range for todays computers, it should be possible to achieve a reasonable accuracy at a reasonable computational cost. Furthermore, both the LHX and the RAH-66 Comanche vehicle requirements were found to be within the predicted feasible region. However, the validation, especially the investigation of the RAH-66 requirements, opens a question as to the capabilities of most conceptual design tools.

The typical contemporary conceptual design tool is a "black box" monolithic code with significant internal constraints hard coded in. The problem with codes of this sort, is that while they are useful for a specific system or class of systems, especially when investigated over a small range, they fall apart when used in broader applications. A reduction in analysis fidelity for some systems, results in a huge increase in capability for analyzing other systems. By using codes similar to the $\mathbf{R_f}$ tool it is possible to increase the number and type of systems investigated using

the RCD methodology. However, most of these "low fidelity" tools lack components to perform economic and other analysis. Since the capabilities provided by these analyses are often part of a system's requirements, it is not possible to fully explore the system requirements space without these capabilities. Therefore, the conversion from the traditional methods to RCD methods indicates that it may be time to investigate developing an entirely new class of conceptual design tools, specifically those with variable fidelity. Only with the addition of these tools and other capabilities will RCD evolve to see its full potential.

CHAPTER VII

CONCLUSIONS & RECOMMENDATIONS

The need to make critical design decisions as early as possible, when design freedom is the greatest and the cost of changes is least, is one that has been the center point of advanced design methodology development for more than a decade. It is with this recognition that the necessity of focusing on the system requirements was developed. Since the specification of requirements greatly decreases the freedom available to the design engineer, it behooves the engineer and the entity setting those requirements to investigate the requirements effect on the final outcome of the system. The UTE was developed to further this goal. However, due to the dependency of this environment on a baseline system, it can only see "evolutionary" changes to a the specific system. Because of this it is necessary to subtly change the approach to the initial design problem.

The basic realization that is made with Requirements Controlled Design is that the requirements define the general "behavior" of the final system, e.g. is it an airplane, a missile, or a laser? Therefore, the requirements are not treated in the same manner as design variables, technologies, or vehicle attributes. Instead requirements are treated as a separate type of variable, one introduced through bifurcation and catastrophe theories: the control variable. The remaining design variables, vehicle attributes, and technology capabilities define the final state/response of the system. Therefore, they are treated in the same manner they always have, as state variables. The benefit of treating requirements as a separate type of variable is that they are no longer tied to a specific system type or baseline concept. Consequently, design freedom is increased. This opens up the ability to prescribe a method to select the system types that are

most appropriate to the current and future requirements and needs.

While designers have been considering what system types are appropriate for years, these evaluations have been heavily dependent on the personal experience of the particular designer. By using the RCD approach, this reliance on personal experience is reduced. The benefit to this is that "out of the box" thinking is encouraged. While a particular organization may be predisposed to producing a certain system type, it is now able to rigorously view, ahead of time, whether or not that system type is most appropriate given its knowledge of current and future needs. RCD also allows the customer to determine, ahead of time, what portions of the requirements (hyper)space are unobtainable at a given technology level. Furthermore, in its ultimate fruition RCD will enable a rapid evaluation of new, innovative, and sometimes random concepts and technologies, thereby pinpointing enabling technologies.

The key to the implementation of a broad-based, requirements-controlled design framework is the recognition of the existence of discontinuous system state boundaries in the requirements (hyper)space. The idea that these boundaries exist is, on some levels, an intuitive one; we already know that it is impractical for a helicopter to fly faster than the speed of sound, but the Concorde has no trouble flying at twice the speed of sound. There is obviously at least one bifurcation in the aerospace systems design space. An achievable method of actually locating these boundaries is possible. While the complexity of most systems negates the ability to analytically find these boundaries, just as it negates the ability to perform fully analytical designs, a numeric method is easily conceivable and, in fact, readily achievable. This thesis and the method contained herein proved just this concept and developed the means of identifying the location of the system state boundaries. Furthermore, with modern meta-modeling techniques, it proved possible to graphically represent the location of the boundaries in any particular part of the requirements (hyper)space.

7.1 Conclusions on the Work Performed

Given that catastrophe theory has been used to describe and understand a variety of inherently complex systems including: markets, animal behavior, and certain aspects of mechanical designs it was reasonably safe to assume that it would be generally applicable to complex systems design. The ongoing trend to bring knowledge forward in the design process, which has focused on understanding the effect of requirements that are encompassed in such methods as UTE and s-Pareto frontiers was the previous state-of-the-art. These methods are extremely powerful and significantly in advance of what was available previously; however, they do not fully leverage the confluence of information technology and the understanding of complex system behavior that has been developed over the years, nor do they fully appreciate how requirements play in complex systems.

To fill this void Requirements Controlled Design was developed. Not only is a new overarching method proposed, justified, and demonstrated, i.e., that system level requirements should be treated as a different type of variable from the typical design/system state variable, but a technique for using this knowledge to further the understanding of complex systems and the effect that requirements have on complex systems was developed, demonstrated and validated. This work has set the foundation for a new set of "equilibrium" design methods. These methods will incorporate the advancements made in this work.

The most basic and ultimately desirable method of investigating the effect of requirements on complex systems is an analytical solution. Unfortunately, previous work with catastrophe theory indicated that a universal analytical solution would, probably, be unattainable. This was not a cause for significant concern. Consider the Navier-Stokes equation, for which no general analytical solution has yet been found. The Navier-Stokes equation is commonly used, either in an approximate form or with numerical solution techniques, in a variety of fields. Further, an entire class

of partial-differential equations, of which the Navier-Stokes equation is part, is widely used without relying upon a universal analytical solution. Even without a general analytical solution to the use of catastrophe theory in design, it would still be beneficial if it were possible to simplify a problem to enable an analytical solution, similar to that of the Blasius solution in fluid dynamics. Unfortunately, for even an extremely simplified, conservative aerospace system, the behavior is such that the analytical solution is as yet unreachable. As was shown this is not a detriment to the use of catastrophe theory principles.

7.1.1 Numerical Exploration of the Requirements Space

Many of the applications of catastrophe theory exist where no good analytical model has ever been created. Furthermore, outside of fanatics, most people understand that a more reformed approach to the use of catastrophe theory allows for the most powerful applications. It is out of this idea that a numerical approach to understanding system level requirements in light of catastrophe theory was developed. Two basic methods for handling this numerical approach were identified and investigated, a grid search and an optimization method.

7.1.1.1 Grid Search Method

The grid search method, which is essentially a brute force way of gaining knowledge about the requirements space, is conceptually very easy to comprehend. The space is divided into a grid and each point on the grid is evaluated. The problem with the grid search is that the combinatorial complexity grows as dimensional powers of the grid density. That means that a 4 dimensional requirements space with a uniform grid density of four points per dimension would require $4^4 = 256$ points. This may not seem to be too severe, but if this relatively sparse grid were extended to the 43 input requirements for the LHX validation example, the total number of evaluations required would have reached $4^{43} = 7.74 \times 10^{25}$ evaluations, which would have been

totally impractical. To add insult to injury is the need to consider the design space in an equilibrium manner.

The need to consider the design space in an equilibrium manner arises from the fact that there are multiple system states at any single point in the requirements space. Therefore, it is necessary to take this into account. If not, there is an extremely high probability that large portions of the feasible region will be missed. Therefore, to properly perform a grid search either a set of the possible states at each of the requirements grid points must be investigated or some sort of optimization routine used. It is immediately obvious that the inclusion of state variables greatly increases the combinatorial complexity, making problems that were impractical completely impossible. The use of an optimization routine internal to the grid search, which was how the analysis of the high-speed strike system was performed, imposes its own set of restrictions. While the total number of function evaluations is significantly decreased, depending on the type of optimization method that is chosen there is, again, a significant risk that a large amount of the feasible space may be missed. Furthermore, if one chooses to implement an optimization based method, it is advantageous to use it entirely in-place of the grid search method.

Even with all of its detriments the grid search method has some very nice benefits for those cases where the number of requirement and state variables is sufficiently reduced and the required resolution is appropriately coarse. The greatest benefit is that a significant amount of knowledge about the requirements (hyper)space is acquired and kept. Furthermore, because of the manner in which the information is gathered it is not necessary to prescribe the locations of the technology boundaries *a priori*. This means that any change in the location of a technology boundary can be added after the fact. Furthermore, it is much simpler to incorporate any change in the uncertainty modeling of the boundaries. Additionally, because of the inherent grid nature of the results, it is very easy to create visualization models.

Since a grid of data points exists at every slice in the space, it is trivial to produce contours of iso-properties. If knowledge is desired between the known slices, simple interpolation or modeling methods suffice. The problem inherent to this is that these models are only as accurate as the fidelity of the grid allows. In an attempt to overcome the limitations of the grid search method another method, one based upon an optimization scheme was devised and investigated.

7.1.1.2 Modified Strength Pareto Method

The identification of the requirements (hyper)space boundaries using the subsystem technology limits by means of an optimization method was undertaken using a modified version of the Strength Pareto Evolutionary Algorithm (SPEA). The modifications were made to the algorithm in order to facilitate the discovery of the Pareto frontier for one or more system types in both the design/state and requirements/control space. This algorithm, dubbed Modified SPEA (MSPEA), was then demonstrated and validated on the U.S. Army's LHX program.

The MSPEA algorithm works by preserving a set of nondominated solutions, where domination is determined in both the parameter/input and response space. This is different from most Pareto seeking multi-objective EAs which are designed to assure nondominance only in the response space. Further, to ensure that a wide variety of points are presented, especially in the parameter/input space, the MSPEA algorithm performs a clustering routine on the external population. Clustering works by determining the vector distance between points, eliminating those that are close together and keeping those that are spread apart, all while keeping to a maximum external population limit. This ensures that all of the points in the external, preserved population are not only unique but spread over as much of the Pareto surface as possible. The two other primary changes made in the MSPEA algorithm are the ability to handle parameters and responses that are both inputs and outputs of the

underlying tool, i.e. the ability to run the tool in an inverse manner, and the ability to allow a separate group of state variables to float freely and not be subject to optimization. This ensures a wide range of states will be incorporated, a key component of an "equilibrium" design process.

The resulting MSPEA tool is capable of finding points on the technology limit induced Pareto front for single or multiple systems. It exhibits good runtime scaling properties with respect to the number of generations, internal, and external population size. Furthermore, in comparison to a fractional grid search method the MSPEA tool is significantly more likely to find a significant portion of the total feasible space. The downside of using the MSPEA tool is the work needed to visualize the results.

Since the MSPEA algorithm only finds pseudo-randomly located points, there is not a grid of results, such as produced by the grid search methods, for use in visualization. This means that there may or may not be any actual results from the MSPEA tool at any arbitrary slice. Neither is it a simple task to produce a prediction through interpolation. Therefore, a more sophisticated modeling method must be employed. This more sophisticated visual modeling method introduces its own complexities and problems. It is necessary to use a visualization modeling method that is capable of dealing with the potentially nonlinear nature of the boundary response in the requirements (hyper)space.

7.1.2 Visualization of the Meta-modeling Results

The task of creating a visualization meta-model, specifically one that is robust, efficient, and capable of growth is an extremely complex undertaking. It is also one that was not solved entirely satisfactorily, especially with respect to quickly displaying the feasible region. The problem encountered with quick predictions of feasibility contours, and specifically of two-dimensional Pareto fronts in n-dimensional spaces is that a large number of evaluations need to be performed to accurately obtain the

visualization data. When this is coupled with a meta-modeling scheme that takes a significant amount of time, hundredths or tenths of a second, to make a single prediction, producing a grid of reasonable density takes a significant amount of time. Ideally, any changes in background variables would be accommodated by the visualization model in real or near-real time. Unfortunately, this goal has, so far, proven unreachable. The primary reason for this is the selection of a meta-modeling routine, the Gaussian Process, which is computationally intensive, coupled with the high dimensionality of the space.

The increasing dimensionality of the requirements (hyper)space, from four dimensions in the early studies contained in this work, to the 43 in the LHX validation case, posed a problem that is common to all meta-modeling methods, i.e. an inherent increase in the prediction error through out the space. This derives from the increased complexity of the space as the dimensionality increases, while the number of sampled points remains fairly constant. Meta-modelers typically attempt to find ways to minimize the number of runs/experiments needed to produce their model in order to make an investigation work, that would otherwise be infeasible. This is exactly how the MSPEA algorithm works. In the case of the boundary discovery method the total amount of space that the boundary encompasses grows much more slowly than the entirety of the space. However, it still grows relatively quickly compared to the rate that is desired by the engineer/decision maker. While the MSPEA algorithm is relatively efficient in handling greater numbers of external Pareto points, i.e. the external population can grow more rapidly than the internal working population, at a ratio of two or three to one; the problem with more information really shows itself in the runtimes of the Gaussian Process meta-model prediction.

The inherent downside of using a Gaussian process is that the time it takes to train and evaluate the process are proportional to the square of the amount of data contained therein. Therefore, as the dimensionality of the requirements (hyper)space

investigated increases and the external population is increased, the runtime of the Gaussian process increases much more quickly. Consequently, it is in the interest of the engineer or decision maker to strike a balance between the amount of data included in the visualization meta-model and the inherent accuracy of the model. An example of this relationship was presented in this work. However, no hard rule has been developed, meaning that there is an effective limit to the number of requirements, and to some extent the number of state variables that can be evaluated at the current time. The benefit here is that the number of dimensions for which analysis is practical will increase more quickly for a MSPEA style method than it will for a grid search style method.

7.2 Suggestions for Future Work

The knowledge garnered from the development, testing, and validation of RCD and the MSPEA based method have led to the identification of several areas for potential future work. Many of these stem from the fact the RCD and the MSPEA are enabling technologies. However, RCD is in need of several new developments before its own full potential can be understood. Even without these developments it is possible to envision some of the future capabilities that are enabled by using RCD.

7.2.1 Needed Developments

Two primary areas of need have been identified through the course of this work. The first and most important is the development of new underlying models. The second is the development of a more capable subsystem assembly process. It is not surprising that these two needs are linked, i.e. you cannot develop a general, automated subsystem assembly process without more capable and general analysis tools.

The typical legacy conceptual design tool is a monolithic "black-box" that was developed to analyze a specific class or sub-class of vehicles. This inherently leads to several problems, not the least of which is that there may be no-one who truly knows

all of the assumptions and simplifications that went into creating the tool. Compounding the problem that tools are designed for a specific type of vehicle, they are often calibrated with specific empirical data and relationships which do not hold for some sub-classes of the given vehicle type.This means that to study a truly diverse set of systems and system types the engineer must collect and, in some manner, integrate several different tools, each with its own assumptions and limitations. Furthermore, since many of these tools were designed with internal technology limitations, which themselves are approximations, they may produce a poor result for a portion of the requirements (hyper)space that is actually feasible. Additionally, many of these legacy tools lack analysis capability for many items that are typically part of the customers decision criterion.

The solution to this problem is the development of a new class of "physics based" conceptual design tools. These tools would initially sacrifice some fidelity in exchange for a more transparent flexible implementation. That is, they would no longer be obscure, black boxes focused upon a specific system, but instead would focus on covering as broad a range of systems as possible. While it is not possible to fully encompass every possible system, by eliminating a large amount of vehicle specific empirical data, it will be possible to greatly increase the range or requirements investigated along with the systems investigated. This will greatly ease the burden of creating requirements Pareto fronts. Additionally, these tools will be able incorporate nontraditional analyses that are often part of the systems requirements; this includes operating economics, environmental concerns, and business case capabilities. Furthermore, an understanding of the tools fidelity needs to be included to facilitate the comparison of results from different system types. This understanding of fidelity leads to another advancement that is needed for the future development of RCD.

Since the initial RCD analysis will be performed on lower fidelity tools, using the subsystem properties to define the state and technology limits, it will be necessary to

include a method of increasing the analysis fidelity over the course of the program. As the program progresses accurate knowledge of the exact location of the technology induced boundaries becomes even more critical. This stems from the decreasing flexibility and increasing cost of change that occur as a program progresses. This can be thought of as programmatic inertia. This equates to the difference between the Maxwell delay convention, where changes are easy and instantaneous, and the perfect delay convention, where changes only happen if there is no capability to remain at the current design choice. Since the accuracy of the technology boundary determination is a matter of the underlying tool's fidelity the knowledge of the boundary should increase in correspondence to the knowledge of the overall system. The primary reason for not using higher fidelity in the initial discovery is that it would be a significantly more computationally intensive task and it may not be possible for all of the systems which need to be investigated. The development of new analysis tools should further the cause of the other needed improvement, the capability to create a diverse set of systems.

The ability to investigate the overall requirements Pareto front is only possible if it can be assured that all pertinent system types are included in the analysis. That is, a strike system analysis would not be complete unless it was possible to analyze both standard aircraft, manned and unmanned, and single use systems such as cruise and ballistic missiles. The problem with the status quo is that each of the systems or system types investigated in RCD must be put together by hand by the analyzing engineer or decision maker. Since the typical matrix of alternatives for a given high level system type may contain tens, hundreds, or even thousands of potential systems this method of analysis quickly becomes impractical. Therefore, a more advanced and partially automated system "assembly" capability needs to be developed. This new capability should contain the option for interaction with the engineer as this can stimulate creativity and solution identification. It is only with these two capabilities,

new analysis tools and advanced system creation, that the full capabilities of RCD can be realized. However, even in its current state RCD will enable development of further new systems design capabilities.

7.2.2 Capabilities Enabled by Requirements Controlled Design

One of the key factors of RCD is the new way in which requirements are viewed. Current methods all view requirements as either goals or simply another variable similar to the system characteristics and technologies. While these methods are often adequate, they limit the flexibility of the design process. The analysis of requirements in the current manner limits the flexibility and understanding of a systems capability. It is by viewing requirements in a different light that great advances are possible. This view is that requirements actually control the behavior of the final system. The advance this brings about is and understanding of the technologies the can be incorporated and the systems which are ultimately chosen. One of the main reasons to use this understanding is that requirements will often change over time and may invalidate the original system chosen.

Requirements controlled design, by viewing systems requirements as control variables, enables an investigation of both system and requirements robustness. That is, the ability of a system to meet changing requirements, and vice versa the ability of the requirements to be fulfilled. The second is especially true if the technologies needed to meet the requirement are currently in development and their capabilities are uncertain. Conversely, if the requirements are beyond what is capable for any of the investigated systems, RCD can help in identifying not only technologies, but also potential new and revolutionary systems. That is RCD can help facilitate revolutionary design programs. Further, if no technology or system can be identified RCD can help identify where the requirements should actually be set.

The combination of all of these capabilities enables what can be called customer

focused design, where systems are chosen and designed, not based on a specific specification, but instead based on a process where the customer's preference and the capabilities of technology are fused to create the most successful program. This melding of markets, systems, and technology is something that is lacking in the current design processes. Using RCD the customer can be informed of the current and future "marginal rates of transformation," i.e. the trade-offs in one or more requirements that are necessary to achieve an improvement in another requirement. Furthermore, by taking into account the customer's preferences it is possible to identify technologies for development.

The identification of technologies for development is possible because of the "equilibrium" approach to design. Since there is no specific baseline vehicle that is perturbed, there is less of a risk of ending up in a poor solution space. Therefore, by operating the MSPEA algorithm to find the requirements induced boundaries in the technology space it is possible to determine the region of subsystem and technology properties that are able to successfully fulfill a set of requirements. This is similar to an "equilibrium" version of the TIF/TIES methodologies. This also makes it possible to determine which system types make the best use of which technologies in improving their performance in specific requirements. Something which is not possible with the current "frozen" design methods.

APPENDIX A

BASIS OF CATASTROPHE THEORY

A basic understanding of the mathematical underpinnings of catastrophe theory is essential if an analytical solution is to be attempted. Additionally, this understanding is helpful in devising an implementation to numerically determine the location of catastrophic boundaries in any control space. Since the "elementary" catastrophes are descriptions of the behavior of critical points of specific functions, an understanding of the classification of critical points is essential.

A.1 Classification of Critical Points

The primary means of classifying critical points of smooth functions is Morse's Lemma. However, in order to properly understand and use Morse's Lemma a succession of terms must be defined.The most basic of these involves the definitions of smooth functions and their critical points. A smooth function is defined as:

Definition 7. Let U be an open subset of $\mathbf{R}^n$. A function $f : U \to \mathbf{R}$ is **smooth** if it has derivatives of arbitrary order [73, p. 2].

Where $\mathbf{R}$ is real space. Smooth functions of an order higher than linear generally posses at least one critical point, which is defined as:

Definition 8. A point $p \in U$ is called a **critical point** of f, if the derivative $Df(p)$ of f at p vanishes [73, p. 2].

Furthermore, critical points can either be isolated or non-isolated. An isolated critical point is defined as:

Definition 9. A critical point $p \in U$ of f is called **isolated** if there is a neighborhood V of p in U such that no other point in V is critical [73, p. 3].

For example Equation 33a contains an isolated critical point at the origin, but Equation 33b does not. The reason for this is that while there is only one critical point for Equation 33a, at the origin; there are an infinite number of critical points in Equation 33b. All of these lie along the y-axis.

$$z = x^2 + y^2 \tag{33a}$$

$$z = x^2 y \tag{33b}$$

Additionally critical points can be classified as either degenerate or non-degenerate. A non-degenerate critical point is defined as:

Definition 10. A critical point $p \in U$ of f is **nondegenerate** if the Hessian matrix $D^2 f(p)$ of f at p is invertible. Otherwise the point p is called **degenerate** [73, p. 4].

For functions of a single variable this means that they will be considered non-degenerate if the derivative $\frac{df}{dx} = 0$ but $\frac{d^2 f}{dx^2} \neq 0$. Thom's "elementary" catastrophes are actually nothing more than a classification of the degenerate critical points of simple monomial and polynomial functions. The seven basic functions are given in Equation 34.

$$x^3, x^4, x^5, x^6, x^3 + y^3, x^3 - xy^2, x^2 + y^4 \tag{34}$$

These functions will later be referred to as the "elementary" catastrophe **germs** [74, p. 18].

Since Morse's Lemma applies, generally, to any number of dimensions it is necessary to consider a n-dimensional real vector space V [73, p. 9]. Sylvester's Law of Inertia applies to these real vector spaces, and is defined as:

Definition 11. Let $F : V \times V \to \mathbf{R}$ be a symmetric bilinear form on an n-dimensional real vector space V. The there are integers r and s with $0 \leq s \leq r \leq n$ that

are uniquely determined and a basis $b_1, \ldots, b_n$ of V such that for any vector $x = x_1 b_1 + \cdots + x_n b_b$ in V

$$F(x,x) = -x_1^2 - \cdots - x_s^2 + x_{s+1}^2 + \cdots + x_r^2 \tag{35}$$

holds, where $r = 0$ means $F = 0$. If $y = y_1 b_1 + \cdots + y_n b_n$ is another vector in V, then

$$F(x,y) = -x_1 y_1 - \cdots - x_s y_s + x_{s+1} y_{s+1} + \cdots + x_r y_r \tag{36}$$

follows from Equation 35 [73, p. 10].

Where a **symmetric bilinear form** on V is a function $F : V \times V \to \mathbf{R}$ satisfying Equation 37 [73, p. 9].

$$F(x,y) = F(y,x) \;\; and \; F(x, y + \alpha z) = F(x,y) + \alpha F(x,z) \tag{37}$$

If one then restricts the symmetric bilinear form on $\mathbf{R}^n$ to the diagonal set, the form becomes quadratic and is defined as:

Definition 12. Let $C = (c_{ij})_{i,j=1,\ldots,n}$ be a real symmetric $n \times n$ matrix. Then the homogeneous polynomial of second degree:

$$q(x) \equiv \sum_{i,j=1}^{n} c_{ij} x_i x_j \tag{38}$$

for $x = (x, \ldots, x_n) \in \mathbf{R}^n$ is called a **quadratic form** on $\mathbf{R}^n$. It is **nondegenerate** if C is invertible and **degenerate** otherwise [73, p. 11].

The quadratic form is inherently second order smooth as shown in Equations 39a & 39b.

$$Dq(x) = 2Cx \tag{39a}$$

$$D^2 q(x) = 2C \tag{39b}$$

Additionally, we will define a linear coordinate transformation as:

Definition 13. A **linear coordinate transformation** on $\mathbf{R}^n$ is a bijective linear map from $\mathbf{R}^n$ to $\mathbf{R}^n$ [73, p. 12].

Where a bijective map is one that is both injective and surjective. These are defined as:

Definition 14. A map $f : X \to Y$ is said to be **injective** if, for all x_1, x_2 such that $f(x_1) = f(x_2)$ we have $x_1 = x_2$ [122].

Definition 15. A map $f : X \to Y$ is said to be **surjective** if, for all $y \in Y$ there exists $x \in X$ such the $f(x) = y$ [122].

Linear algebra provides that a linear map can be represented by a real $n \times n$ matrix T such that Equation 40 holds.

$$\tau : \mathbf{R}^n \to \mathbf{R}^n$$
$$\tau(x) = Tx \; for \; x \in \mathbf{R}^n \tag{40}$$

Therefore, if T is invertible then τ is a linear coordinate transformation. Using Equation 38, we see that a quadratic form q is actually a linear map on $\mathbf{R}^n$. Additionally using a linear coordinate transformation on a quadratic form, produces a quadratic form [73, p. 12].

Theorem 1. Classification of Quadratic Forms on $\mathbf{R}^n$ Let q be a quadratic form on $\mathbf{R}^n$. Then there are integers r and s with $0 \leq s \leq r \leq n$ that are uniquely determined and a linear coordinate transformation τ of $\mathbf{R}^n$ such that

$$q \circ \tau(x) = -x_1^2 - \cdots - x_s^2 + x_{s+1}^2 + \cdots + x_r^2 \tag{41}$$

holds for $x = (x_1, \ldots, x_n) \in \mathbf{R}^n$, where $r = 0$ means $q = 0$. In particular the quadratic form q is non-degenerate exactly where $r = n$ [73, p. 14].

The proof of this is provided by Domenico and Hayes [73, pp. 14-16].

However, because of the nature of the germs listed in Equation 34, it is necessary to study not just general linear coordinate transformations, but smooth ones. A smooth coordinate transformation is defined as:

Definition 16. A map $F: U \to \mathbf{R}^m$ is **smooth** f each component $F_j \equiv pr_j \circ F$ is smooth, $j = 1, \ldots, m$, where $pr_j : \mathbf{R}^m \to \mathbf{R}$ is the projections $(y_1, \ldots, y_n) \to y_j$ [73, p.18].

With smooth maps defined it is now possible to define a specific map called a diffeomorphism:

Definition 17. Let V be open in $\mathbf{R}^n$. A bijection $\phi : U \to V$ is a **diffeomorphism** if both ϕ and ϕ^{-1} are smooth [73, p. 18].

Most smooth maps are not diffeomophisms; however, if a local diffeomorphism exists about the origin it is possible to apply Morse's Lemma. A local diffeomorphism is defined as:

Definition 18. A smooth map $\phi : U \to \mathbf{R}^n$ is a **local diffeomorphism at point** p in U if an open neighborhood V of p in U exists such that $\phi(V)$ is open in $\mathbf{R}^n$ and $V \to \phi(V)$, $x \to \phi(x)$, is a diffeomorphism. A **local inverse of** ϕ **at** $\phi(p)$ is a smooth map $\psi : W \to \mathbf{R}^n$ defined on an open neighborhood W of $\phi(p)$ satisfying $\psi(\phi(x)) = x$ for $x \in \phi^{-1}(W) \cap V$. A local diffeomorphism as a point $p \in U$ is also called a smooth coordinate transformation at p [73, p. 19].

With the definitions of a smooth coordinate transformation at p and a local diffeomorphism it is now possible to express Morse's Lemma:

Lemma 1. Let f vanish at $0 \in U$. The origin is a nondegenerate critical point of f if and only if a local diffeomorphism ψ at 0 exists with $\psi(0) = 0$ such that

$$f(\psi(y)) = -y_1^2 - \cdots - y_s^2 + y_{s+1}^2 + \cdots + y_n^2 \tag{42}$$

holds at the origin, where s denotes the index of f at 0 [73, p. 24].

How does Morse's Lemma help us? All of the functions listed in Equation 34 have degenerate isolated critical points at the origin. However by perturbing the monomial and polynomial equations it is possible to produce nondegenerate critical points. These perturbations are the keys to the development of both the control and state variables and the "elementary" catastrophes.

A.2 Control & State Variables

Imagine the function given in Equation 43.

$$\mathbf{Y} = f(\mathbf{X}) \tag{43}$$

Where $\mathbf{X} = (x_1, \ldots, x_n) \in \mathbf{R}^n$. This function has critical points which meet the following criteria.

$$Df(\mathbf{X}) = 0 \tag{44}$$

Now imagine a small perturbation of this function, as shown in Equation 45.

$$\hat{\mathbf{Y}} = f(\mathbf{X}) + \mathbf{C} \cdot g(\mathbf{X}) \tag{45}$$

Where $\mathbf{C} = (c_1, \ldots, c_l) \in \mathbf{R}^l$. The critical points of $\hat{\mathbf{Y}}$ in the functional space $\mathbf{X}$ are found at:

$$Df(\mathbf{X}) + \mathbf{C} \cdot Dg(\mathbf{X}) = 0 \tag{46}$$

Therefore the position of the critical points in $\mathbf{R}^n$, are actually functions of a variable in $\mathbf{R}^l$. i.e.

$$\mathbf{X} = h(\mathbf{C}) \tag{47}$$

Now define $\mathbf{Y}$ as the **state** of the system; therefore, since the setting of $\mathbf{X}$ determines the value of $\mathbf{Y}$ the $\mathbf{X}$ are now called the SVs, which are defined in Definition 4, on page 32.

Given the perturbation described in Equation 45, the state of $\hat{\mathbf{Y}}$ is now determined by $\mathbf{X}$ which is itself dependent on $\mathbf{C}$. Therefore it can be said that **C controls**

X. Henceforth, the components **C** will be known as the CVs, which are defined in Definition 3, on page 32.

We can now return to the functional relationship given in Equation 43, this can be replaced by one of the germs given in Equation 34, as shown in Equation 48.

$$y = f(x) = x^3 \tag{48}$$

Using Definitions 8, 9, & 10 it is possible to determine that Equation 48 posses an isolated, degenerate critical point at $x = 0$. There are many potential perturbations to this function. A family of these perturbations is called an **unfolding**. However, the subset of the unfoldings that is the most interesting is the **versal** set, defined as:

Definition 19. The unfolding $F(\mathbf{X}, \mathbf{C})$ is called **versal** if it describes, qualitatively, the complete spectrum of all possible behaviors of the system f under perturbation [73, p. 40].

In the cases where there are a limited number of versal unfoldings, this unfolding is referred to as a universal unfolding. For functions of a low order there are a limited number of versal unfoldings. In the case of Equation 48, the universal unfolding is given in Equation 49.

$$F(x, c) \equiv x^3 + cx \tag{49}$$

The presence of one control variable in the universal unfolding of x^3 indicates that it is a function of order 3, and **codimension** of 1; further, the fact that it contains one state variable indicates that the function has a **corank** of 1. The corank, codimension and universal unfolding for the remainder of the germs given in Equation 34 are given in Table 18. The classification of the critical points for both the germs and their universal unfoldings produce the seven or ten (some literature considers the "duals" to be separate) "elementary" catastrophes described by Thom.

Table 18: Universal Unfoldings for Germs of Corank ≤ 2 and Codimension ≤ 4, Including Duals

germ	**corank**	**codimension**	**unfolding**
$[x^3]$	1	1	$[x^3 + cx]$
$\pm[x^4]$	1	2	$\pm[x^4 - c_1x^2 + c_2x]$
$[x^5]$	1	3	$[x^5 + c_1x^3 + c_2x^2 + c_3x]$
$\pm[x^6]$	1	4	$\pm[x^6 + c_1x^4 + c_2x^3 + c_3x^2 + c_4x]$
$[x^3 - xy^2]$	2	3	$[x^3 - xy^2 + c_1(x^2 + y^2) - c_2x - c_3y]$
$[x^3 + y^3]$	2	3	$[x^3 + y^3 + c_1xy - c_2x - c_3y]$
$\pm[x^2y + y^4]$	2	4	$\pm[x^2y + y^4 + c_1x^2 + c_2y^2 - c_3x - c_4y]$

A.3 Elementary Catastrophes

The functional relationship described by Equation 48 has a single critical point at $x = 0$. When classified it becomes evident that $Df(0) = 0$ and $D^2f(0) = 0$ meaning that the critical point is isolated and degenerate. Further, the universal unfolding for Equation 48 is shown both in Equation 49 and Table 18. This unfolding adds exactly one CV dimension to the single SV dimension in the original germ. This indicates that the unfolding has a codimension and corank of 1 and 1 respectively. Further the nature and location of the critical point changes. For the trivial case where $c = 0$, the critical point remains identical to that in the germ. However, for conditions where $c \leq 0$ & $c \geq 0$ the behavior changes dramatically. To further investigate this behavior, one needs to look at the behavior of the first derivative of F, given in Equation 50

$$\frac{\partial F(x,c)}{\partial x} = 3x^2 + c \tag{50}$$

Here we see why c is called a control variable, to determine the critical points of F the value of x is determined or "controlled" by the value of c_1. Setting Equation 50 $= 0$ and solving for x we obtain Equation 51:

$$x = \pm\sqrt{-\frac{c}{3}} \tag{51}$$

This produces an interesting phenomena. The critical point exists on the real axis if and only if $c \leq 0$. For $c > 0$ the critical points exist away from the real axis,

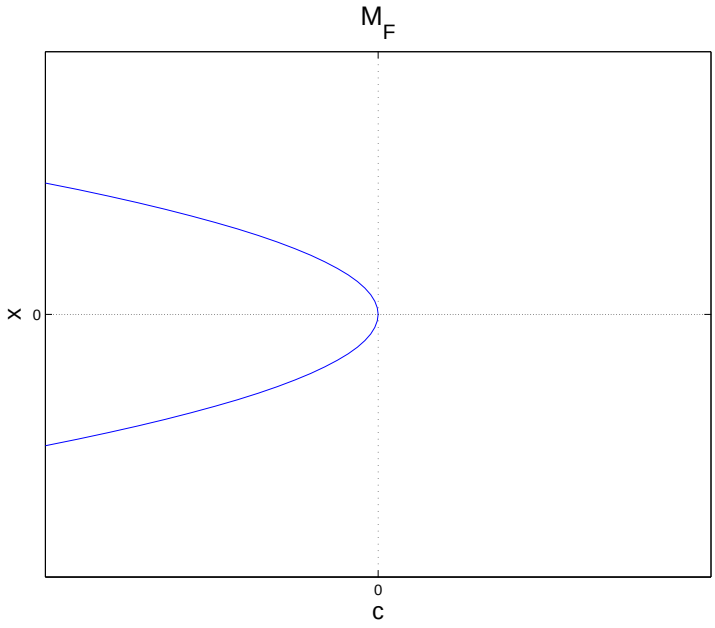


Figure 75: Cubic Germ Catastrophe Surface (M_F) [73, p. 41]

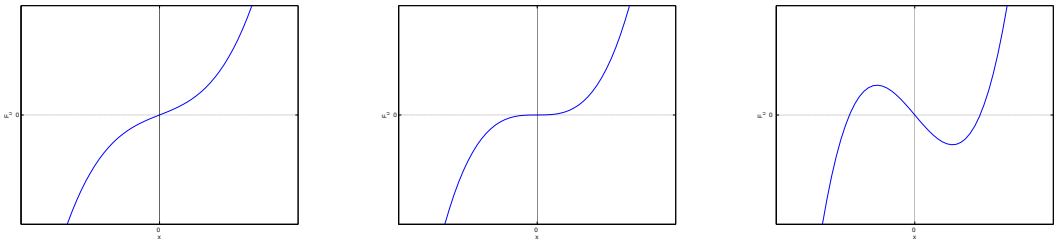
in the complex plane plane. For the trivial case of $c = 0$ there is only one critical point. However for $c < 0$ there exist two isolated, non-degenerate critical points, one minimum and one maximum, located at the points given by Equation 51. The set of all of the critical points of all partial functions $F_c(x)\ of\ F(x, c)$ is called the **catastrophe surface** [73, p. 42]. In the case of Equation 49, this is the parabola given in Equation 52.

$$M_F = \{(x, c) : 3x^2 + c = 0\} \tag{52}$$

This surface is shown in Figure 75. Three different representations of F_c are given in Figure 76.

The behavior of the critical points of x^3 is the most basic of the elementary catastrophes, the **fold** catastrophe. Each of the unfoldings given in Table 18 has an associated catastrophe. These catastrophes are shown in Table 19.

As the order and codimensionality of the problem increases the behavior of the "elementary" catastrophes becomes more interesting. The universal unfolding of x^4



(a) $c > 0$ (b) $c = 0$ (c) $c < 0$

Figure 76: Partial Functions F_c of F [73, p. 42]

Table 19: The Elementary Catastrophes

germ	**codimension**	**unfolding**	**Catastrophe**
$[x^3]$	1	$[x^3 + cx]$	Fold
$\pm[x^4]$	2	$\pm[x^4 - c_1x^2 + c_2x]$	Cusp
$[x^5]$	3	$[x^5 + c_1x^3 + c_2x^2 + c_3x]$	Swallowtail
$\pm[x^6]$	4	$\pm[x^6 + c_1x^4 + c_2x^3 + c_3x^2 + c_4x]$	Butterfly
$[x^3 - xy^2]$	3	$[x^3 - xy^2 + c_1\,(x^2 + y^2) - c_2x - c_3y]$	Elliptic Umbilic
$[x^3 + y^3]$	3	$[x^3 + y^3 + c_1xy - c_2x - c_3y]$	Hyperbolic Umbilic
$\pm[x^2y + y^4]$	4	$\pm[x^2y + y^4 + c_1x^2 + c_2y^2 - c_3x - c_4y]$	Parabolic Umbilic

produces the **cusp** catastrophe, shown in Figure 77. It is in the cusp catastrophe that all of the common properties of an "elementary" catastrophe are present. These include [74, p. 21]:

- Bimodality: The system can exist in two or more distinct equilibrium states [123, p. 158].
- Inaccessibility: The system has unstable equilibrium states [123, p. 158].
- Sudden jumps: A small change in a control parameter produces a large change in the system state, i.e. a catastrophe [123, p. 159].
- Hysteresis: Transition between states occurs at different values of the control variables [74, p. 21]. The system is not fully reversible [123, p. 161].
- Divergence: A small change in a control parameter leads to a large change in

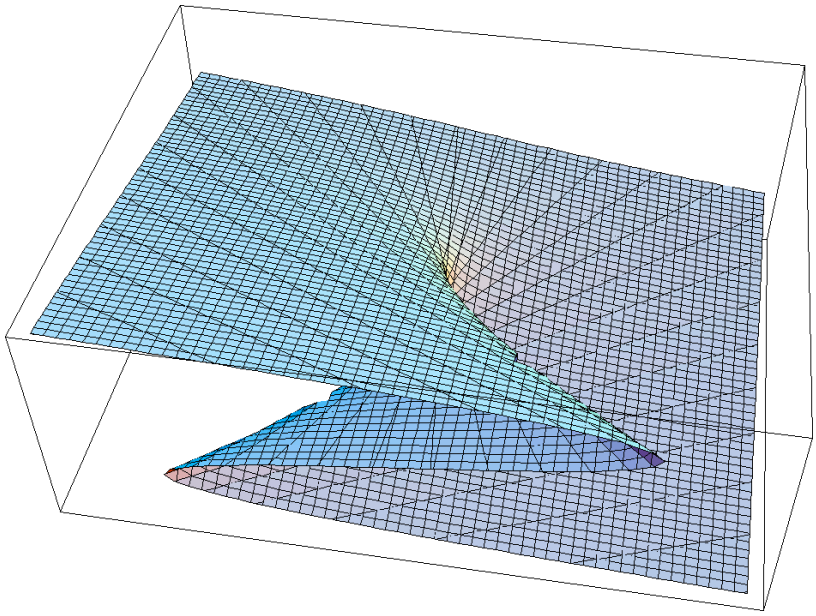


Figure 77: The Cusp Catastrophe Surface (M_F)

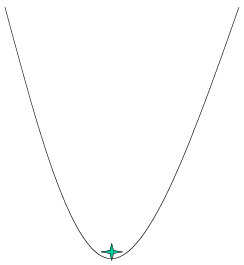
the final state; however no sudden jumps occur (not present for the fold) [123, p. 160].

These properties are also known as *catastrophe flags* [123, pp. 157-166]. The existence and discovery of these flags is important in the cases where a proper analytical model may not be available.

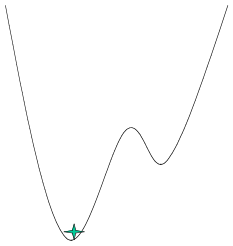
A.4 Hysteresis in State Transition

One of the more interesting properties of the "elementary" catastrophes is the presence of hysteresis in state transition. The exact behavior of the state hysteresis is based upon several factors. The most important of these is called the **delay convention** [123, pp. 142-144]. This delay determines at what point the system transitions from one state to another. Figure 78 shows an example of hysteresis based upon what is called a "perfect" delay [124, p. 84].

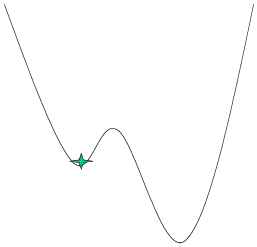
The initial entry state is shown in Figure 78(a), the system is in the optimum state for a given set of CVs as there is only a single solution available. A second



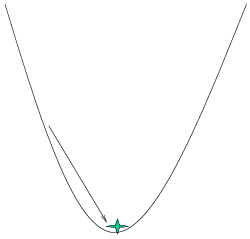
(a) Initial Control Variable Setting, Single Global Minimum



(b) Second Control Variable Setting, Multiple Minima, Initial Minimum is Global



(c) Third Control Variable Setting, Multiple Minima, Initial Minimum is Local



(d) Fourth Control Variable Setting, Original Minimum Disappears

Figure 78: Example of Hysteresis in Two Control Variables, Single State Variable [125]

setting of one CV, shown in Figure 78(b), illustrates a later point. At this time two possible solution states exist; however, the initial point remains the global optimum. A further change in one CV, shown in Figure 78(c), results in a change in the initial minimum from the global to a local minimum. However, because the system exhibits a "perfect" delay it does not transition immediately. It is only in the final change of one CV, shown in Figure 78(d), when the original minimum no longer exists, that the system state finally changes.

Since the delay convention is mathematically a continuum it is possible to have a less than perfect delay. As the delay decreases the amount of hysteresis present in the state transition also decreases. Ultimately there exists a system where no delay exists, and the system automatically transitions to the global minimum. This system, the opposite of the perfect delay convention, posses what is known as the **Maxwell convention** [123, pp. 143-144].

A.5 Extension of Catastrophe Theory to Higher Orders

One of the limitations of pure catastrophe theory, as proposed by Thom, is that as the dimensionality and co-dimensionality increase the number of potential catastrophes quickly approaches infinity. For co-dimensions > 5 and coranks > 2 the number of catastrophe forms is infinite [74, p. 19]. Up to a combined dimensionality of eleven it is possible to classify a finite number of catastrophe families. At higher dimensionalities these too become infinite in number. These infinite families of catastrophes are often referred to as "non-elementary" or "generalized" catastrophes [74, p. 19]. It is in this set of higher-order modes that the links to chaos theory and fractal geometry are found [74, p. 19]. This rapid approach to infinity leads one to question the general applicability of Catastrophe Theory to the design of highly complex systems.

APPENDIX B

GRID SEARCH METHOD

The simplest and most straightforward method of investigating the requirements (hyper)space and discovering the technology induced feasibility boundaries is the simple grid search. To perform a grid search there is no need to predetermine the technology boundaries, as they are applied after the fact. Furthermore, the results from the grid search provide information about the entire requirements (hyper)space, not just the location of the boundaries. This is particularly useful if the location of the boundaries changes unexpectedly. However, all of this information comes at a cost, combinatorial complexity. The grid search example contained in this chapter was performed using a notional hypersonic fighter-concept.

B.1 Grid Search Example System

The notional hypersonic strike-fighter concept investigated was the same system used to evaluate the viability of the requirements for a hypersonic strike-fighter [6]. The notional hypersonic cruise missile requirements were created especially for this particular study. However, they were representative of systems currently being evaluated in the missile design field. Table 20 lists the four requirements investigated and their upper and lower bounds. The requirements investigated were each chosen to represent

Table 20: Requirements for the Grid Search Study [7]

Requirement	Lower Bound	Upper Bound
Time-to-Target	15 min	60 min
Mission Radius	750 nm	3,000 nm
Payload Weight	750 lbs.	9,000 lbs.
Gross Weight	1,500 lbs.	100,000 lbs.

a different category:

- Time: Time-to-target
- Range: Mission Radius
- Payload: Payload/Warhead Weight
- Size: Gross Weight

The grid was a basic six-level grid with additional points. This required 1536 runs for each system, resulting in a total of 3072 runs. Three particular subsystem level properties were tracked:

- Required I_{sp}
- Required System $\frac{T}{W}$
- Required Cruise Mach Number

These represent propulsion and airframe technology metrics, and specific limits for each will determine the requirements space sections for which each solution is feasible. Other system state variables were adjusted using a SQP optimizer to achieve the desired requirements settings.

B.2 Demonstration of Grid Search Method

Since the analysis was performed over a grid, a significant amount of knowledge was obtained about the requirements hyperspace. This makes it possible to not only determine the location of a technology limit with respect to the requirements, but also to show the curvature of a metric throughout the requirements hyperspace. Unfortunately, because the requirements hyperspace chosen for study was four-dimensional, only planar slices can be easily shown with a display technique similar to that used by Ashby.

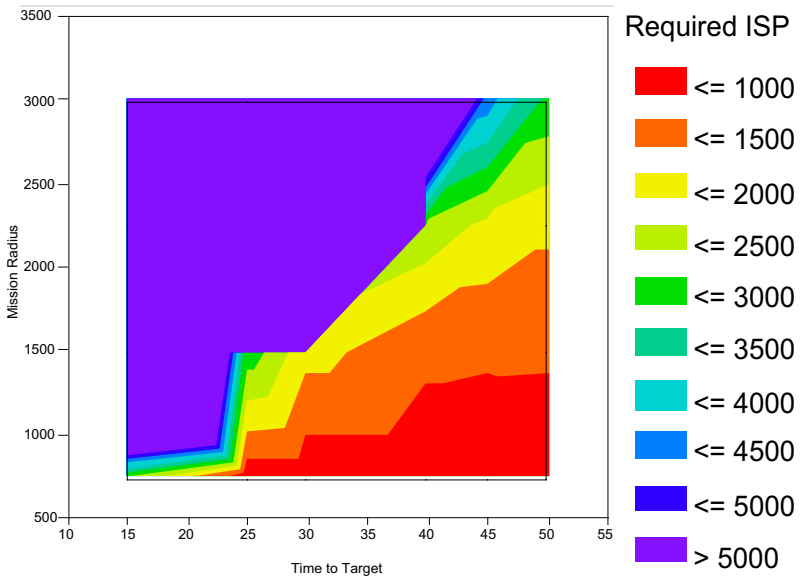


Figure 79: Hypersonic Strike-fighter Required I_{sp} Contours [7]

B.2.1 Required Specific Impulse

The first technology topography investigated was that for the required I_{sp}. To simplify the visualization of the hyperspace surface, the contours shown are for varying time-to-target and mission radius. For the hypersonic strike-fighter the payload was set at 2,250 lbs., and the gross weight at 50,000 lbs. The required I_{sp} topography for the hypersonic strike-fighter is shown in Figure 79.

The cruise missile, is obviously a smaller system, with a lower payload and lower maximum launch weight. In the required I_{sp} study the payload was set a 750 lbs. and the launch weight at 3,000 lbs. The topography for the cruise missile is shown in Figure 80.

While it may seem inappropriate to compare different size vehicles, with different payloads, this is not always the case. The strike-fighter is sized to deliver a single 2,000 lb JDAM type bomb. The cruise missile is designed to deliver a multipurpose warhead from the Navy standard vertical launch system (VLS). The maximum gross weight of both systems allows for growth with respect to their launch system, providing similar

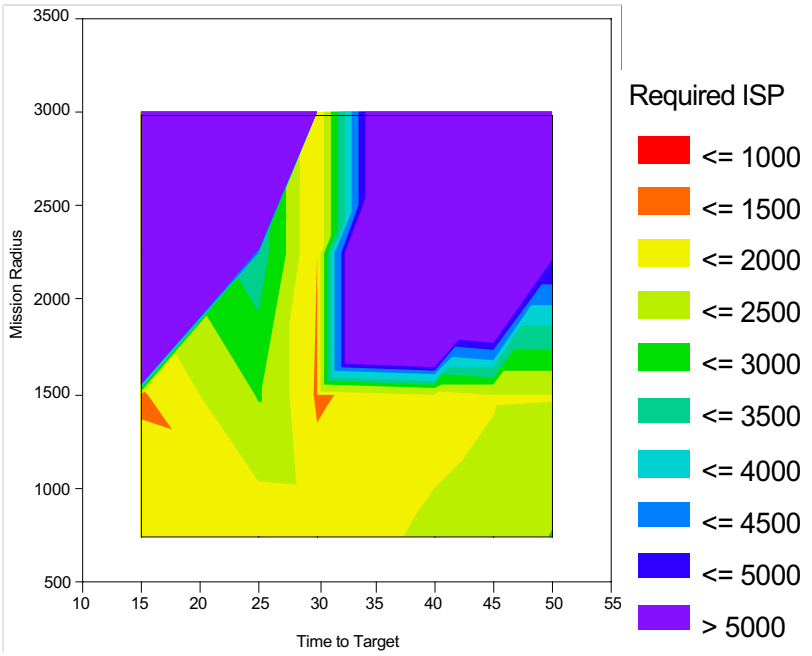


Figure 80: Cruise Missile Required I_{sp} Contours [7]

kinetic energy to their respective payloads on delivery.

The topography shown in Figures 79 and 80 indicates that both systems preform better for the shorter range, longer time to target missions. It is also obvious that the required I_{sp} increases rather abruptly for both systems. This indicates that the control the time-to-target and mission radius have on the system I_{sp} is significantly higher than linear in order.

B.2.2 Required Thrust to Weight Ratio

Similar results for the required system thrust to weight ratio are presented. Again they are displayed for varying time-to-target and mission radius requirements. The payload and gross weight limits are the same as those used to display the required I_{sp}. Figure 81 shows the topography for the hypersonic strike-fighter, and Figure 82 shows the topography for the cruise missile.

While the topography for the strike-fighter shows an increasing requirement for system thrust to weight ratio as the time-to-target is decreased and the mission radius

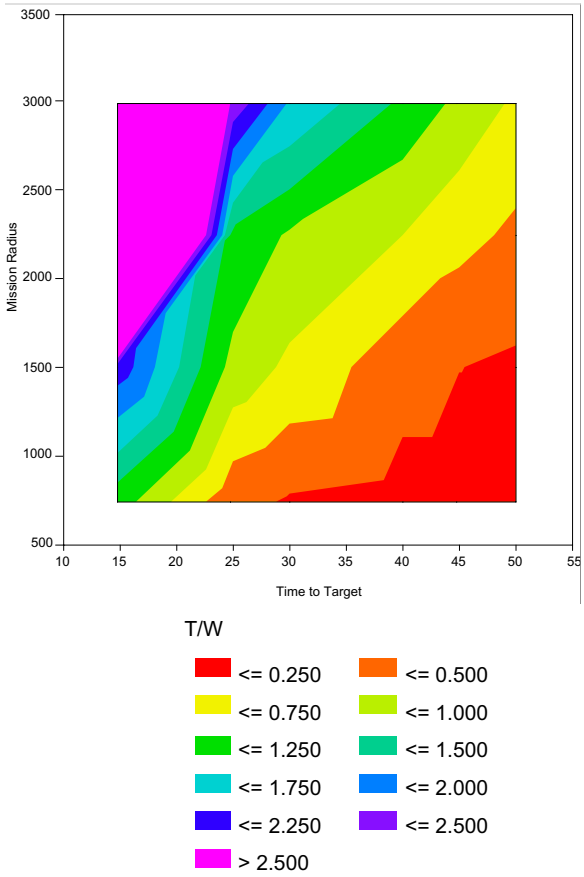


Figure 81: Hypersonic Strike-fighter Required $\frac{T}{W}$ Contours [7]

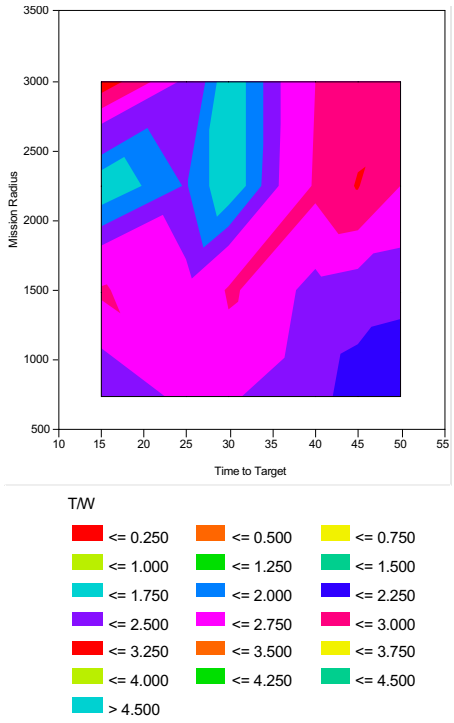


Figure 82: Cruise Missile Required $\frac{T}{W}$ Contours [7]

is increased, the effect that the requirements have on the cruise missile is less apparent. The general trends derive from the need to increase the mission block speed as the mission radius is increased while holding the time-to-target requirement constant. The same effect is seen for a decreasing time-to-target and a fixed mission radius.

B.2.3 Required Cruise Mach Number

Similar results for the required system cruise Mach number are presented. Again they are displayed for varying time-to-target and mission radius requirements. The payload and gross weight limits are the same as those used to display the required I_{sp}. Figure 83 shows the topography for the hypersonic strike-fighter, and Figure 84 shows the topography for the cruise missile.

The topography for the required cruise Mach number displays the same influences as the required thrust-to-weight ratio topography figures. However, in this case the curvature for the cruise missile is significantly more prevalent.

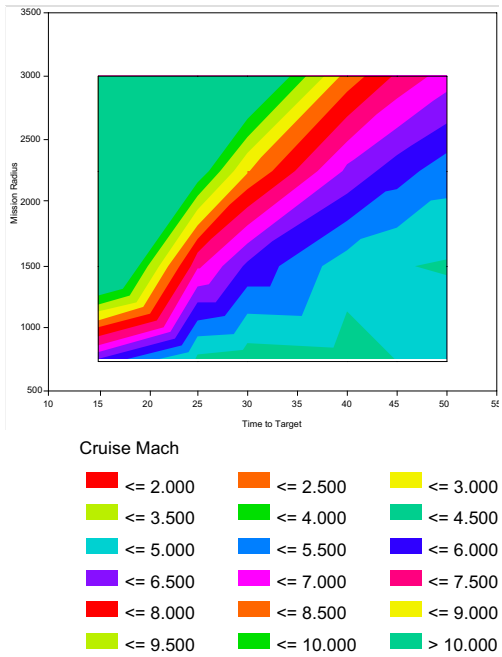


Figure 83: Hypersonic Strike-fighter Required Cruise Mach Number Contours [7]

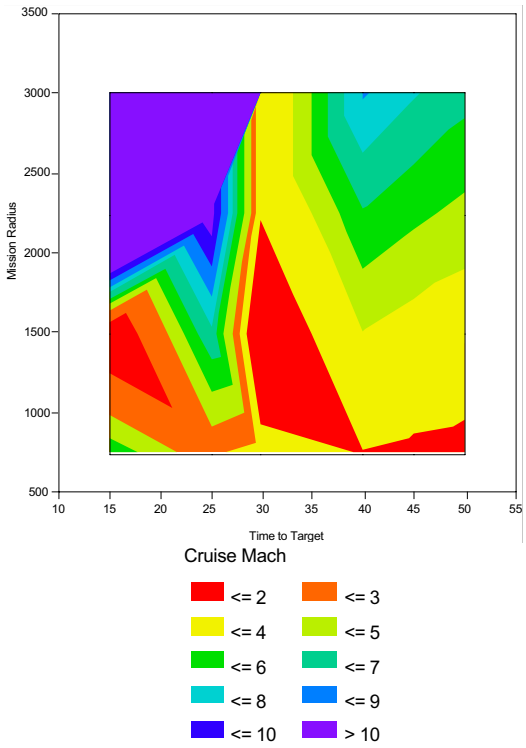


Figure 84: Cruise Missile Required Cruise Mach Number Contours [7]

Table 21: Subsystem Technology Limits

Limit	Fighter	Missile
Cruise I_{sp} (sec)	1800	1800
System $\frac{T}{W}$	1.25	3.0
Cruise Mach	6	6

B.2.4 Display of Technology Boundaries

While knowledge of topography of the system requirements hyperspace with respect to subsystem level metrics is beneficial, it does not directly indicate the location of the system state catastrophic boundaries in the requirements hyperspace. To visualize these using the grid search method it is necessary to determine the limits as they apply to the particular study. These limits may correspond to the current or predicted future state of the art, or a physical limit of a particular technology or cycle. In the case of the strike-fighter vs. the cruise missile example the limits are listed in Table 21.

The difference in system thrust-to-weight originates from the fact that the structural mass fraction for a reusable aircraft will be higher than that for a single use missile. Additionally, the strike-fighter is required to carry its entire propulsion system throughout the entire mission, while the cruise missile can dispose of its booster stage. The maximum cruise Mach limit and the maximum I_{sp} correspond to a hydrocarbon ramjet cruise propulsion system. Since the purpose of this study was to compare the viable portions of the requirements hyperspace of the hypersonic strike-fighter vs. the cruise missile, the limits for each subsystem metric were placed on the same chart.

Figure 85 shows the limit contour for the required cruise I_{sp} for both systems. Of note is the "trough of feasibility" for the cruise missile which extends throughout the range of mission radii at the 30 minute time-to-target value. It is also interesting that, for longer time-to-target requirements on the shorter mission radii, the I_{sp} required

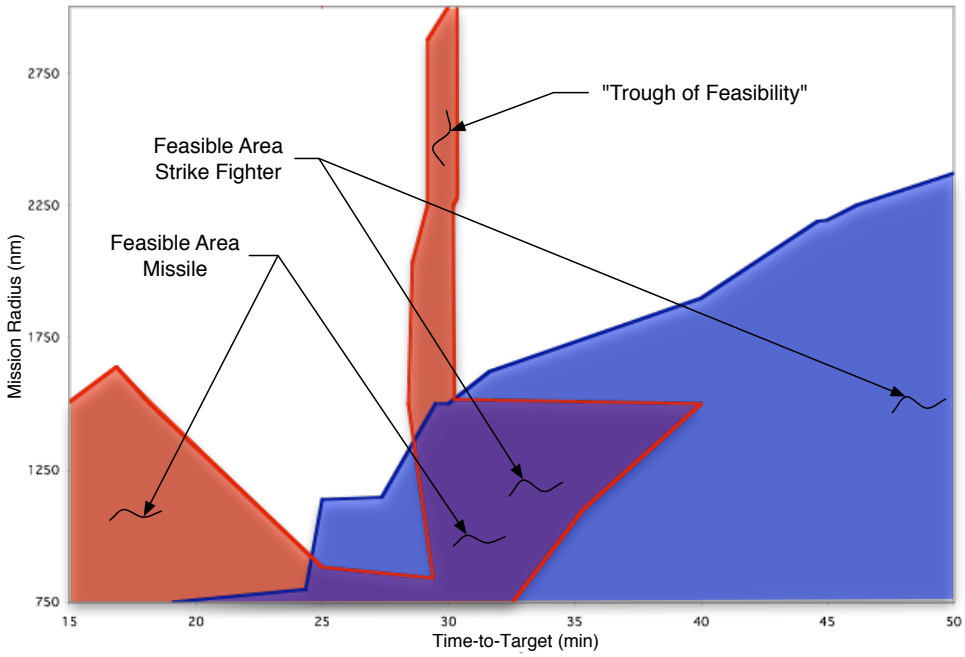


Figure 85: Required Cruise I_{sp} Limit, 1,800 seconds

for the missile is above the 1,800 second limit. This is due to the time dependent nature of fuel burn.

Figure 86 shows similar limits for the required system $\frac{T}{W}$. The limit line plot for the $\frac{T}{W}$ clearly shows what the topographic plots implied, that the $\frac{T}{W}$ technology limit for the cruise missile would not be a factor in determining the region of the requirements hyperspace for which the cruise missile is a feasible solution.

Figure 87 shows similar limits for the required cruise Mach number. The cruise Mach number limit of 6 imposes a considerably more stringent requirements limit than does the $\frac{T}{W}$. In this case, because of the greater acceleration rate of the cruise missile, the portion of the requirements hyperspace that can be satisfied by the cruise missile is significantly larger than the portion that can be satisfied by the strike-fighter.

Individual technology limits are useful to understand; however, it is the combined limits that determine the portion of the requirements hyperspace for which each

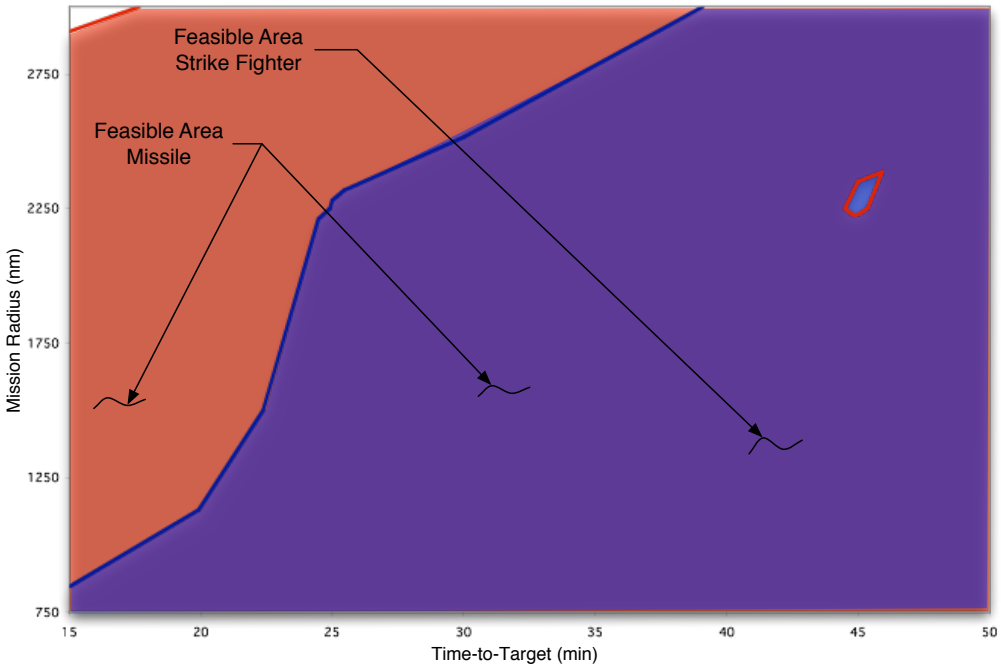


Figure 86: Required Cruise $\frac{T}{W}$ Limit

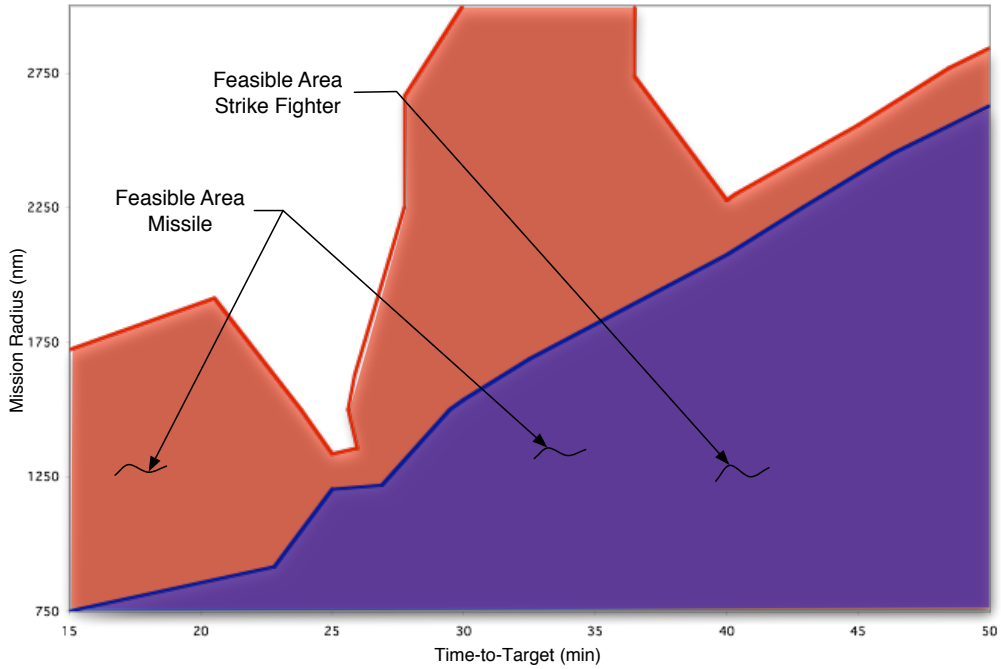


Figure 87: Required Cruise Mach Number

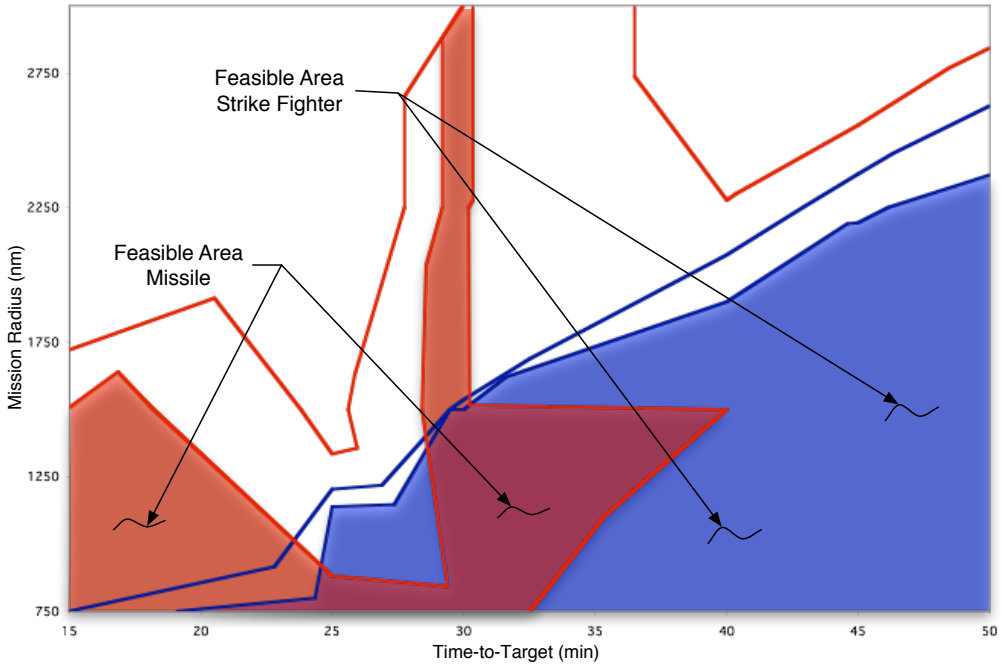


Figure 88: Overlaid Technology Limit Contours

system is a viable solution. It is relatively straightforward to obtain the individual limits from the grid data. To obtain the combined, "Pareto", limit for each system requires a more complicated routine. Figure 88 shows all of the technology limits for both systems overlaid.

The inherent problem with overlaying technology limits is that the plot becomes hard to read. However, Figure 88 clearly shows the presence of regions where none, one, or both of the systems are feasible. Depending on the requirements value, the previous experience of the engineer, and the predicted changes in requirements, either the strike-fighter or the missile could be chosen as a solution to a high-speed, time critical strike system.

APPENDIX C

BASIC GENETIC ALGORITHM METHOD

One of the simplest of the global optimization methods that can be used in the technology boundary discovery process is the simple Evolutionary/Genetic Algorithm (EA/GA). In order to test the capability of a basic GA to discover the range of the technology induced feasibility boundaries in the requirements (hyper)space, a relatively simple test case, involving a cruise propulsion system for a high-speed air-breathing vehicle system, was performed. The number of requirements, technology boundaries, and state variables investigated was relatively small, five of each. However, the total dimensionality of the space was increased significantly over that of the grid search method examined in Appendix B. This stems from the fact that investigating only the requirements allows for the possibility of nonunique solutions. The total number of dimensions that the underlying tool was allowed to vary in was increased from four to ten. This, coupled with an increase in the resolution of each dimension, increased the combinatorial complexity significantly. Making a grid search computationally infeasible.

C.1 Basic Evolutionary Algorithm Discovery of the Catastrophic Boundaries

The use of a basic GA to discover the catastrophic boundaries was performed by the author and the results were presented in 2002 [107]. The primary purposes of this particular study were to demonstrate the effectiveness of a GA in discovering the locations of both the individual and Pareto technology boundaries, and to further demonstrate the validity of the hierarchical decomposition method. The notional

Table 22: Notional High-Speed, Cruise Propulsion System Requirements/Control Variables [107]

Requirements	Lower Bound	Upper Bound
Mach#	2.0	6.0
I_{SP} (sec)	500	4500
Fn_{SP} (lblbsec)	50	250
Inlet Area (ft^2)	0.1	10
Fuel Type	H_2	JP

Table 23: Technological Limits Imposed Upon the High-speed, Cruise Propulsion System [107]

Technology Limit	Value	Unit
η_{inlet}	0.80 - 0.95 [126]	
$\eta_{combustor}$	0.95	
η_{nozzle}	0.99	
T_4	4500 & 5000	$^\circ$ R
T_3	2900	$^\circ$ R

system chosen was a high-speed air-breathing, cruise propulsion system. This is the same type of system that would be used to propel the hypersonic strike-fighter evaluated in the previous section. The control variables for this system are given in Table 22. To simplify the problem, the JP and H_2 powered ramjets were treated as separate system types. However, there is no reason that the systems could not have been treated as a single system type, with fuel mixture as a variable. The other requirements are either the state variables or control variables for the higher-level, vehicle system.

The nature of EAs in general and GAs in particular, necessitates that the system property technology boundaries be identified prior to using the GA. Because of the simple cycle of ramjet and scramjet engines, these are often efficiency and temperature limits. The particular limits chosen for this study are given in Table 23.

The ramjet/scramjet cycle code used for this study was RAMSCRAM from NASA Glenn [127]. Furthermore, because RAMSCRAM is a legacy code, some of the CVs

Table 24: Input/Output Status of Requirements & Limit Variables [107]

Requirement	Status	Variable Name
I_{sp}	Output	ISP
Mach Number	Input	AMO
Fn $_{SP}$	Output	SPF
Fuel Type	Input	multiple
Inlet Area	Input	ADES1
Technology Limit	**Status**	**Variable Name**
η_{inlet}	Input	AKD1
$\eta_{combustor}$	Input	AKD3
η_{nozzle}	Input	CS
T_4	Output	T4
T_3	Output	T3

Table 25: Additional Design/State Variables Used in the Genetic Algorithm [107]

State Variable	Range	Variable Name
Altitude	30,000 - 150,000 ft	ALT
Nozzle Velocity Coefficient	0.9 - 1.0	CV
Normal Shock Mach#	1.1 - 6.0	AMD2
Nozzle Exit Area	1 - 10 ft^2	ANOZZ
Equivalency Ratio	0.95 - 1.05	PHI

and SVs are input variables and some are responses. The status of both the requirements and the technology limits, plus their variable names are listed in Table 24. Additionally, there were several other design variables that were allowed to float freely. This was done so that a closed, complete ramjet cycle would be guaranteed for each run. These variables, ranges, and variable names are listed in Table 25.

C.1.1 Properties of the Genetic Algorithm

The GA used in this test case implemented a tournament selection routine, a 40% crossover ratio, where whole gene sections were swapped, and a 30% single bit, random mutation rate. Figure 89 contains a flowchart showing the algorithmic implementation of the GA. The GA was run for 120 iterations before termination. Several repetitions were performed showing that the population had stabilized prior to termination. Therefore, the inclusion of an auto termination routine may provide runtime benefits.

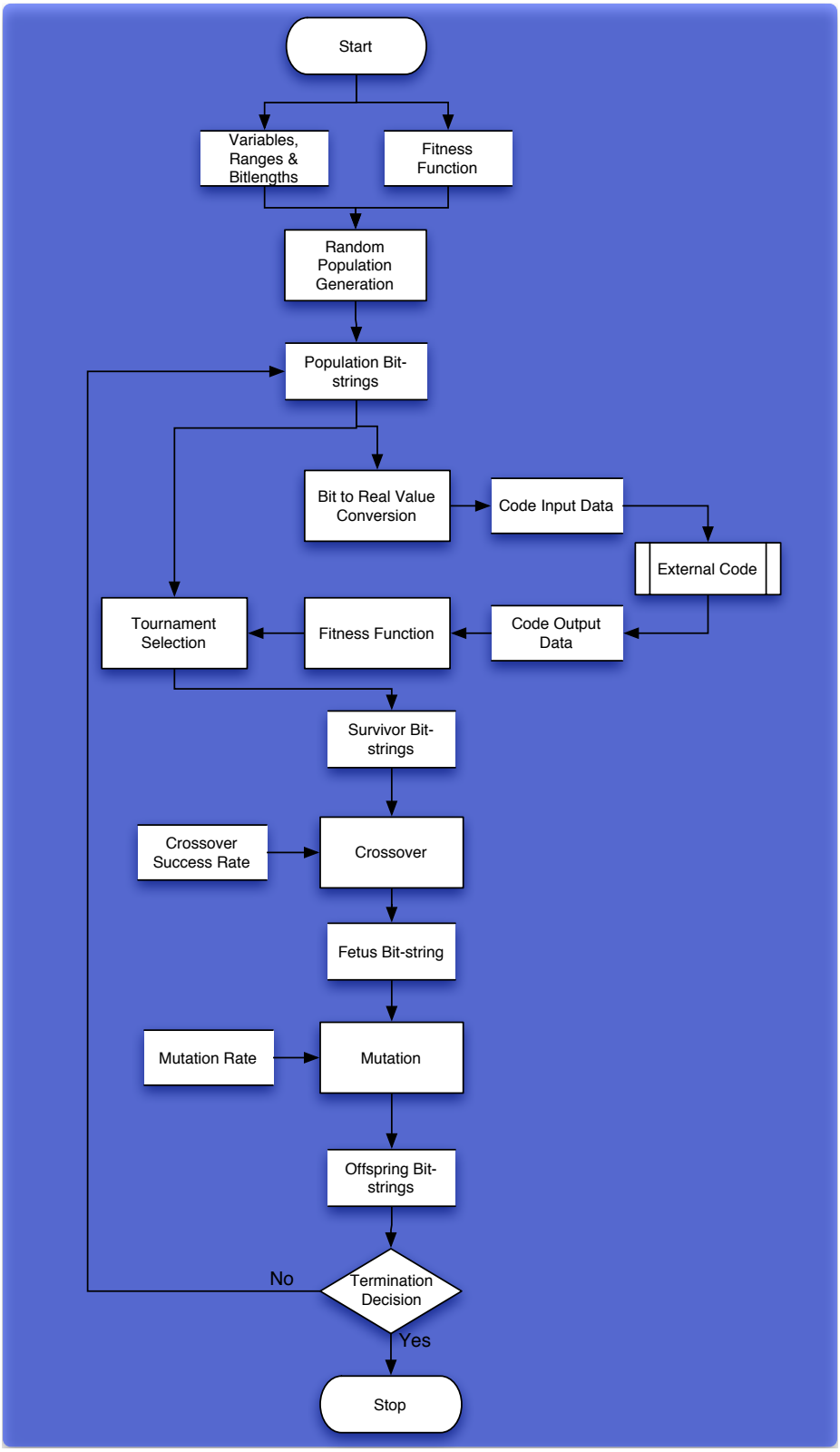


Figure 89: Algorithmic Flow Chart for Simple Genetic Algorithm

Table 26: Fixed Control Variable Settings for the Individual Technology Boundaries

Control Variable	Value
Inlet Area	1 sqft.
Mass Flow	70 lbf/lbm/sec

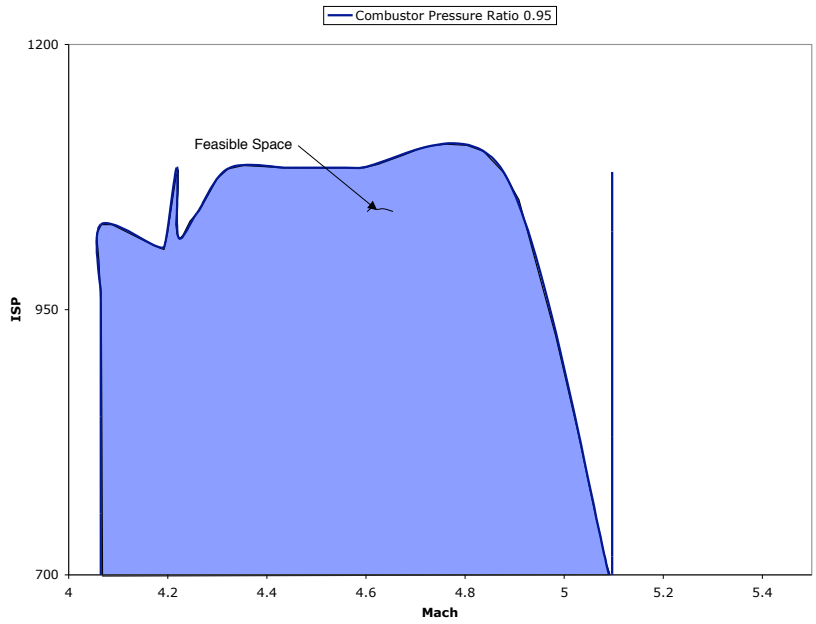
C.1.2 Results of the GA Discovery of Technological Boundaries

The GA successfully determined the presence of individual technology limits for the JP fueled ramjet. These limits were determined in a four-dimensional hyper-space. Therefore, visualization is a difficult matter. In order to minimize the amount of data presented to the reader, only a slice of the space is shown. Through reduction of the data, it was determined that the inlet area and specific thrust presented themselves as more correlated; therefore, the axes chosen for presentation were Mach and I_{sp}, fixing the other two requirements. The specific values for inlet area and mass flow rate, for both the H_2 and JP are given in Table 26.

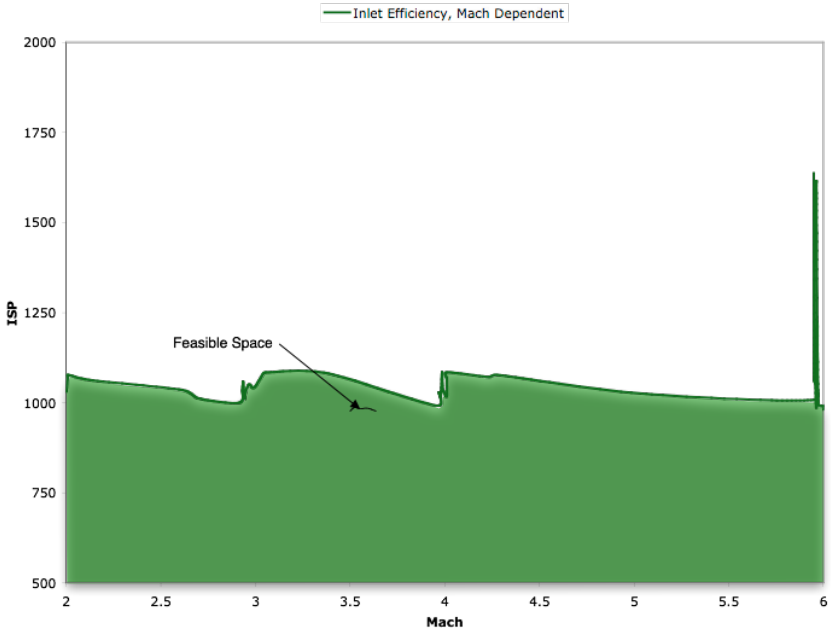
C.1.2.1 JP Fueled Ramjet

The individual technology based limit results for the JP fuel propulsion system are shown in Figures 90 & 91. Figure 90(a) shows the 0.95 combustor pressure ratio limit, Figure 90(b), the Mach number dependent inlet efficiency limit, Figure 91(a), the combustor inlet wall temperature limit, and Figure 91(b), the combustor exit wall temperature limit. Combining all of the boundaries, produces the technology limit for the entire JP fueled system. This combined limit is shown in Figure 92.

It is interesting to note that the technology limit boundaries track almost identically at a constant I_{sp} for the lower Mach numbers; however the nozzle efficiency seems to decrease the I_{sp} as the Mach number decreases. Additionally, because no work is done on the flow during compression, the location of the temperature limits in the requirements hyper-space are solely a function of one CV, Mach number. Figure 92 shows that for a T_4 limit of 4500 °R, the maximum Mach limit occurs around Mach 4.6. An increase in the technological capability of the engine to a T_4 of 5000

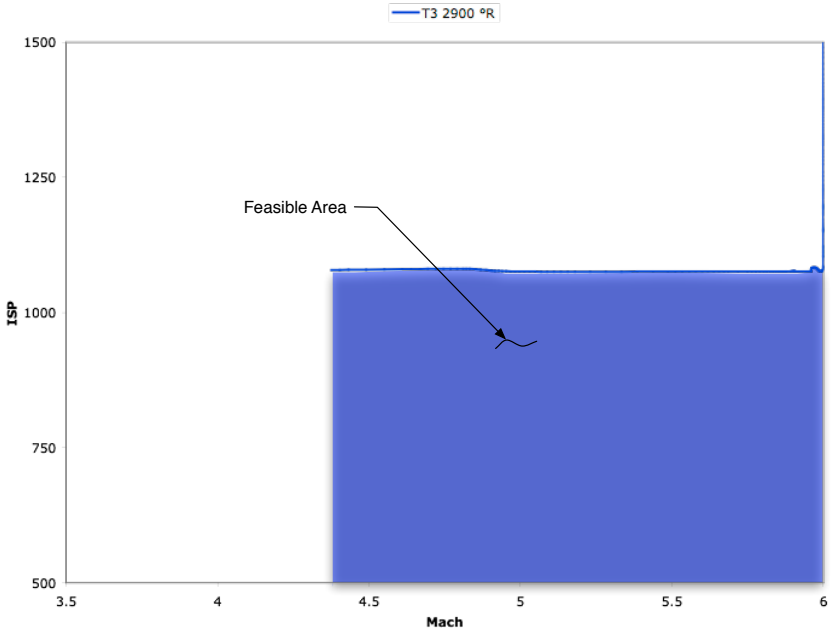


(a) Combustor Pressure Ratio

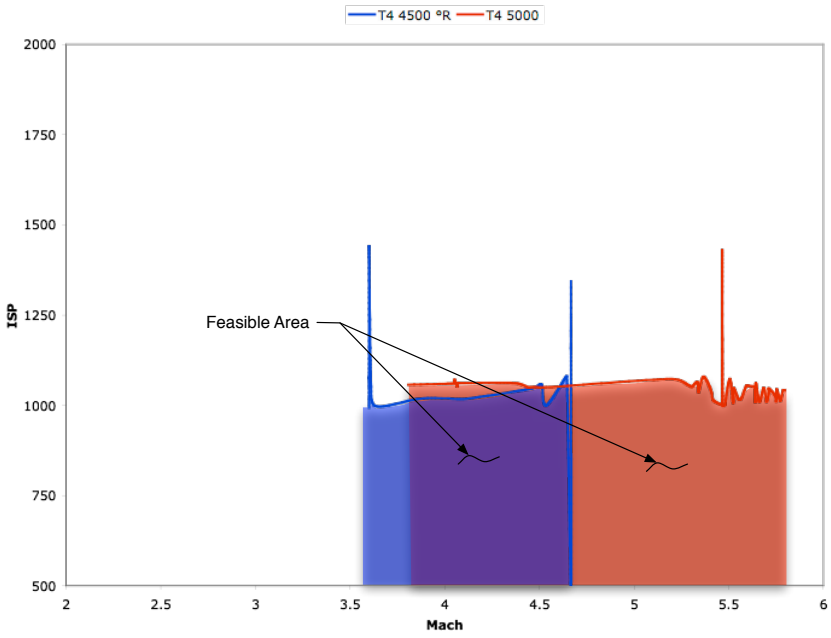


(b) Inlet Efficiency

Figure 90: Individual Limits for JP Fueled Propulsion System Requirements Space



(a) Combustor Inlet Temperature



(b) Combustor Exit Temperature

Figure 91: Individual Limits for JP Fueled Propulsion System Requirements Space Cont.

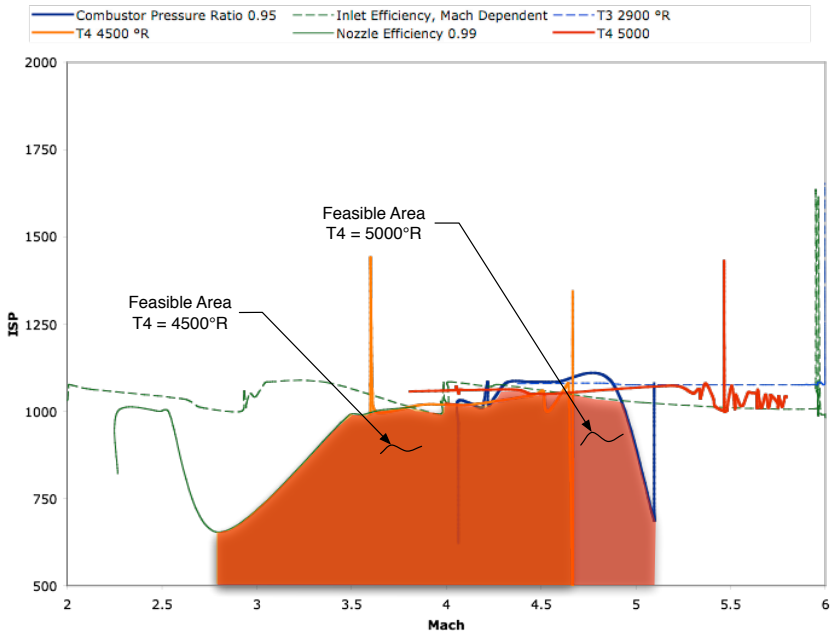


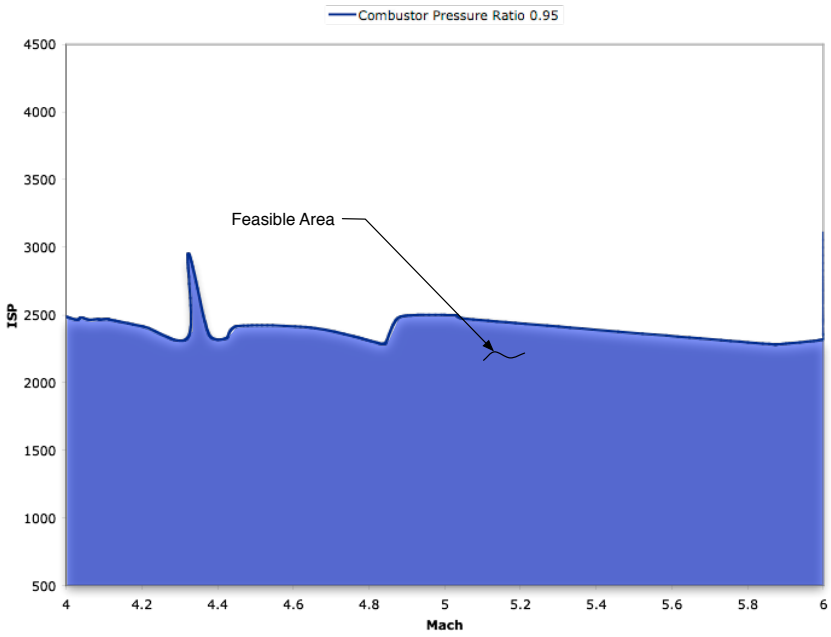
Figure 92: *JP* Fueled Propulsion System Requirements Space, Mach vs. I_{sp} [107]

°R, which may be provided through active cooling of the combustor and nozzle walls, increases the maximum Mach number to greater than Mach 6.

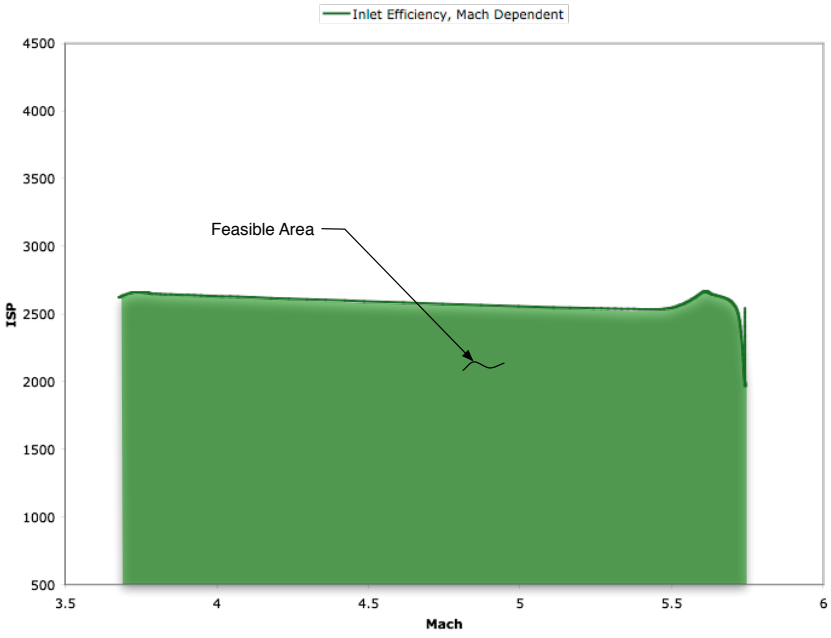
C.1.2.2 H_2 Fueled Ramjet

The GA was successful in identifying the individual requirements boundaries for the hydrogen fueled ramjet. The maximum I_{sp} range for the H_2 ramjet was significantly higher than for the *JP* fueled ramjet. This is to be expected. If volumetric I_{sp} had been included as one of the requirements the hydrogen ramjet would have fared differently. The individual technology limits are shown in Figures 93 & 94. Figure 93(a) shows the 0.95 combustor pressure ratio limit, Figure 93(b), the Mach number dependent inlet efficiency limit, Figure 94(a), the nozzle efficiency limit, and Figure 94(b), the combustor exit wall temperature limit. The combined limit results for the H_2 fueled ramjet are presented in Figure 95, on page 202.

Again the results were essentially similar, with the nozzle efficiency limit decreasing the I_{sp} boundary at lower Mach numbers. The T_4 limit of 4500 °R produces a

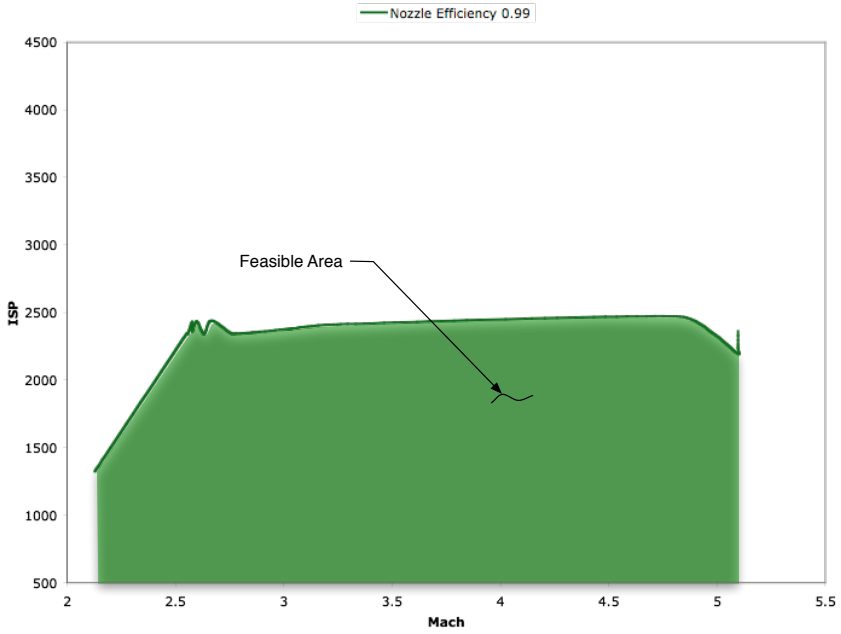


(a) Combustor Pressure Ratio

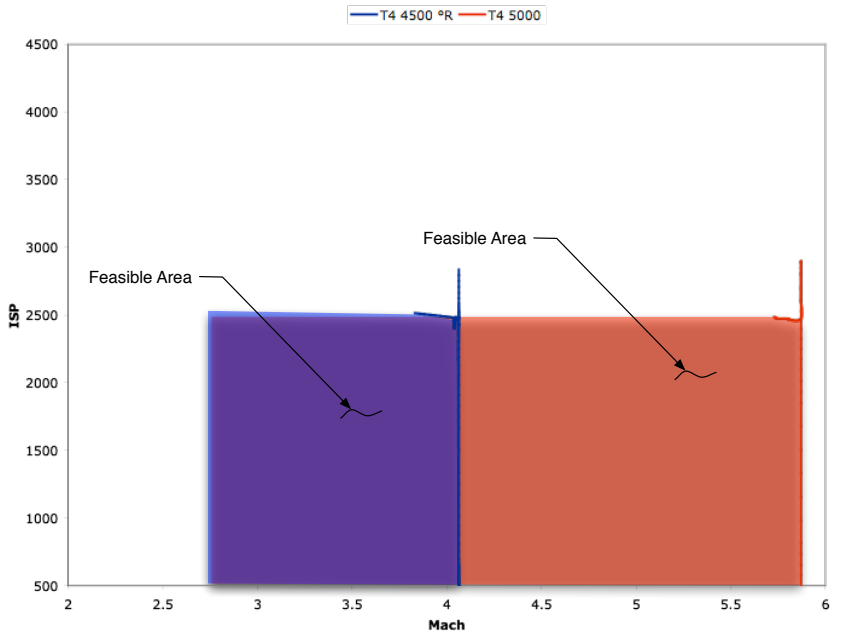


(b) Inlet Efficiency

Figure 93: Individual Limits for H_2 Fueled Propulsion System Requirements Space



(a) Nozzle Efficiency



(b) Combustor Exit Temperature

Figure 94: Individual Limits for H_2 Fueled Propulsion System Requirements Space Cont.

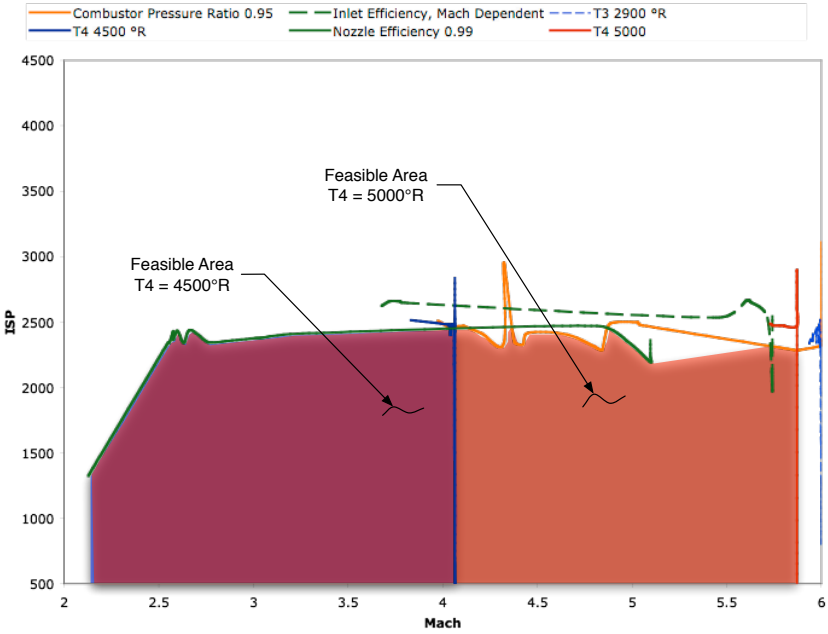


Figure 95: H_2 Fueled Propulsion System Requirements Space, Mach vs. I_{sp} [107]

maximum Mach limit of around 4.1, with the limit increased to a T_4 of 5000 °R, this increases to Mach 5.9. Furthermore, it is of interest to note that the combustor efficiency limit did not produce a minimum Mach number limit, further suggesting that the results in the JP fueled ramjet case were an aberration. Additionally, in neither case did the T_3 maximum temperature limit produce an effect in the requirements range studied.

C.1.2.3 Combined Requirements Boundaries

The success of the GA at determining the individual technology boundaries, while less than ideal, precipitates the desire to investigate the combined technology limit for each candidate system. The combination of the boundaries for all of the system types can be called the requirements Pareto front. That is, at the given level of technology being studied, the requirements cannot be set to more stringent values than those that are along the front. For this case, the planar slice of the requirements hyperspace that is displayed was taken with the fixed control variable values shown

Table 27: Fixed Control Variable Settings for the Combined Technology Boundaries

Control Variable	Value
Inlet Area	1 sqft.
Mass Flow	100 lbf/lbm/sec

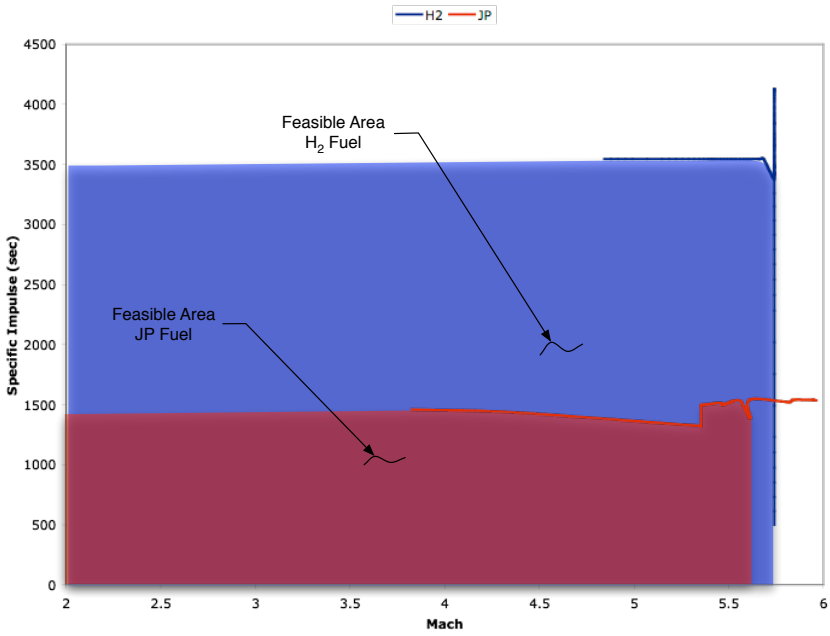


Figure 96: Requirements Space, Combined Boundaries, Mach vs. I_{sp} [107]

in Table 27. The results for both the JP and H_2 fueled propulsion systems are shown in Figure 96.

The GA was generally successful in finding the Pareto limit for the ramjet in the requirements hyper-space. However, the trends associated with the nozzle efficiency that were visible at the lower Mach numbers in both Figures 92 & 95, are not present in Figure 96. It seems that the GA did not produce any results below Mach 3.75 for the JP fueled ramjet and below Mach 4.8 for the H_2 fueled ramjet. Further study is required to properly ascertain the reasons for this; however, it is likely that fewer low Mach cases satisfied the GA goals in the Pareto front analysis, particularly the Mach limit. Even with this problem, the GA proved quite capable of determining the requirements Pareto front.

Table 28: Comparison of the Computational Effort for Different Requirements Boundaries Discovery Methods [107]

Method	Function Calls	Total Function Call CPU Time
Grid	$3.6\text{x}10^{16}$	57 million years
Individual GA	90,750	2.36 hours
Combined GA	9,075	7.6 minutes

C.1.3 Computational Benefits of the GA vs. Grid Search Method

The poor scaling of the grid search base method was mentioned and discussed in the previous section. This particular study focused on the comparison of the computational requirements that would have been required to achieve the same level of fidelity for the combined technology boundaries. In the case of the RAMSCRAM code, each function call took approximately 0.05 seconds to accomplish. There is additional overhead associated with both methods; however, for the end result the effect of this overhead is negligible. The GA was run using eight bits to represent each variable, producing the equivalent of a 32 level grid space. Because of the nature of the RAMSCRAM code there were eleven input variables. The resulting number of function calls and the project CPU time are given in Table 28 The obvious conclusion is that, in the case of the propulsion system space, the use of the grid based search is completely infeasible. Even if the grid search only dealt with the six CVs and technology limits that were inputs, the number of function calls would have been $\sim 1.1 \times 10^9$; taking ~ 1.7 years to complete. This is still combinatorially unacceptable.

C.1.4 Downsides to the Basic GA Approach

The use of the simple GA, while promising, identified several downsides that must be addressed. The first and foremost is the tendency of the GA to focus on a single solution point; some runs produce a broad front, others focus on a few or occasionally a single point. The most obvious solution for this is to vary the objective to ensure a broader front of the technology limits is observed. This can be achieved by using a

multi-objective EA. The second downside is the lack of grid based data. Since the GA is an inherently random process, even though it is discretized, there is no guarantee that data exists to take planar slices of the requirements hyperspace. The solution to this is to create a meta-model of the technology limit across the entire hyperspace.

The combination of a multi-objective EA and meta-model of the boundary data acquired by the EA would ensure that a sufficient number of points along the boundary are available to produce the meta-model. Further, once the meta-model is produced it is straightforward and computationally efficient to produce an Ashby chart of the available systems in any two dimensions of the requirements hyperspace. When combined with an event based interface, automatically updated dynamic contours would be available to the user.

APPENDIX D

MSPEA PROGRAM SKELETON

D.1 Main Program

D.1.1 JAVA_SPEA Main Class

JAVA_SPEA.java

```
//
//  JAVA_SPEA.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Fri Sep 19 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
// This is the Master code for the MSPEA algorithm implementation with the RF_Spreadsheet
// JAVA library
//
// GOD HELP ME!!!!!!!
//
import java.util.*;
import edu.asdl.design.ea.basic.*;
import edu.asdl.design.ea.spea.*;
import edu.asdl.design.ea.spea.Generations;
import edu.asdl.design.ea.spea.ParetoPopulation;
import edu.asdl.design.ea.spea.ExternalParetoPopulation;
import edu.asdl.design.ea.spea.ParetoIndividual;
import edu.asdl.design.rf.*;
import java.io.*;

public class JAVA_SPEA {

    Runtime nowhere = Runtime.getRuntime();

    //public RF_Objective rf_Objective;
    public ParetoPopulation Pop;
    public Generations evolver;
    String Arguments[], input_filename, output_filename,
        objective_name = "RF_Objective";
    int max_generations =50, number_indiv, max_pop, max_threads, ext_pop;
    boolean set_gen = false, set_input = false, set_output = false, set_pop = false,
        set_objective = false, set_threads = false, set_ext = false;
    String Inputs[][], Outputs[][], Tracked[][], Free[][], Xs[], Ys[], Fs[], Ts[],
        Xtype[], Ftype[], Vehicle_Type, Types[][];
    double xlimits[][], ylimits[], tlimits[][], flimits[][];
    int xbits[], fbits[];
    boolean internal_out = false;

    public JAVA_SPEA() {
    }

    public void start() {
        //
        // Initialize by reading Command line arguments
        //
        // The arguments include number of generation, number of internal population members,
        // maximum number of external population members, maximum number of objective threads,
        // a flag to write-out internal populaiton data, input file name, and output filename
        // suffix. Code is removed to save space.
        //
        //
        //
        // Handle Input File & create population with objectives
        //
        inputFileReader();
        populationCreate();
        objectiveCreate();
        //
        // Initialize evolver with requisit stuff
        //
        evolver = new Generations(Pop);
        evolver.setMaxNumGenerations(max_generations);
        //
        // Start Evolver
        //
        if (set_threads) {
            evolver.putMaxThreads(max_threads);
        }
```

```
        if (set_ext) {
            evolver.setExternalPopulationSize(ext_pop);
            System.out.println("External Population size: " +ext_pop);
        }
        evolver.startThread();
        try {
            evolver.GA_thread.join();
        } catch (InterruptedException ie) {
        }
        // Evolver Finished
        //
        ExternalParetoPopulation externPop = evolver.getExternalPopulation();
        //
        // Write out External Population Data
        double xfinal[][] = externPop.getXs();
        OutputWriter Xout = new OutputWriter(("X_Final."+output_filename), Xs, xfinal);
        double yfinal[][] = externPop.getYs();
        OutputWriter Yout = new OutputWriter(("Y_Final."+output_filename), Ys, yfinal);
        double tfinal[][] = externPop.getTs();
        OutputWriter Tout = new OutputWriter(("T_Final."+output_filename), Ts, tfinal);
        double ffinal[][] = externPop.getFs();
        OutputWriter Fout = new OutputWriter(("F_Final."+output_filename), Fs, ffinal);
        // If chosen write out Internal Population Data
        if (internal_out) {
            ParetoPopulation internPop = evolver.getInternalPopulation();
            double xfinalin[][] = internPop.getXs();
            OutputWriter Xout2 = new OutputWriter(("X_Final.Internal."+output_filename), Xs,
                                                  xfinalin);
            double yfinalin[][] = internPop.getYs();
            OutputWriter Yout2 = new OutputWriter(("Y_Final.Internal."+output_filename), Ys,
                                                  yfinalin);
            double tfinalin[][] = internPop.getTs();
            OutputWriter Tout2 = new OutputWriter(("T_Final.Internal."+output_filename), Ts,
                                                  tfinalin);
            double ffinalin[][] = internPop.getFs();
            OutputWriter Fout2 = new OutputWriter(("F_Final.Internal."+output_filename), Fs,
                                                  ffinalin);
        }
        //
        int generation_num[] = externPop.getGenerationNumbers();
        OutputWriter Genout = new OutputWriter(("External_Gen."+output_filename), generation_num);
        System.out.println("Done");
    }
    //
    void populationCreate() {
        short numgenes = (short)(Xs.length + Fs.length);
        short numbits[] = new short[numgenes];
        int totalbits = 0;
        for (int ii=0; ii<xbits.length; ii++) {
            numbits[ii] = (short)xbits[ii];
            totalbits += numbits[ii];
        }
        for (int jj=xbits.length; jj<numgenes; jj++) {
            numbits[jj] = (short)fbits[jj-xbits.length];
            totalbits += numbits[jj];
        }
        if (set_pop) {
            number_indiv = max_pop;
        } else {
            number_indiv = totalbits * 3;
        }
        Pop = new ParetoPopulation(number_indiv, numgenes, numbits);
    }
    //
    void objectiveCreate() {
        for (int ii=0; ii<number_indiv; ii++) {
            ParetoIndividual tempIndiv = (ParetoIndividual)Pop.getIndividual(ii);
            RF_Objective tmpObj = new RF_Objective(Vehicle_Type, Xs, Xtype, Ys, Ts, Fs, Ftype,
                                                   xlimits, ylimits, tlimits, flimits, tempIndiv);
            tempIndiv.createFitness(tmpObj);
            tempIndiv.initialChromosome();
            tempIndiv.update();
            Pop.putIndividual(ii, tempIndiv);
        }
        Pop.setNumberOptParam((short)Xs.length);
        Pop.setNumberTracked((short)Ts.length);
    }
    //
    void inputFileReader() {
        BufferedReader buffread;
        StringBuffer input_buffer;
        //
        //
        // Code to read and parse the input file
        //
        //
        InputSplit();
    }
    //
    void InputSplit() {
```

```
        Xs = new String[Inputs.length];
        Xtype = new String[Inputs.length];
        xbits = new int[Inputs.length];
        xlimits = new double[Inputs.length][2];
        Ys = new String[Outputs.length];
        ylimits = new double[Outputs.length];
        Ts = new String[Tracked.length];
        tlimits = new double[Tracked.length][2];
        Fs = new String[Free.length];
        Ftype = new String[Free.length];
        fbits = new int[Free.length];
        flimits = new double[Free.length][2];
        //
        for (int ii=0; ii< Inputs.length; ii++) {
            Xs[ii] = Inputs[ii][0];
            Xtype[ii] = Inputs[ii][1];
            try {
            xlimits[ii][0] = Double.parseDouble(Inputs[ii][2]);
            xlimits[ii][1] = Double.parseDouble(Inputs[ii][3]);
            xbits[ii] = Integer.parseInt(Inputs[ii][4]);
            } catch (NumberFormatException nfe) {
            }
        }
        //
        for (int ii=0; ii< Outputs.length; ii++) {
            Ys[ii] = Outputs[ii][0];
            try {
                ylimits[ii] = Double.parseDouble(Outputs[ii][1]);
            } catch (NumberFormatException nfe) {
            }
        }
        //
        for (int ii=0; ii< Tracked.length; ii++) {
            Ts[ii] = Tracked[ii][0];
            try {
                tlimits[ii][0] = Double.parseDouble(Tracked[ii][1]);
                tlimits[ii][1] = Double.parseDouble(Tracked[ii][1]);
            } catch (NumberFormatException nfe) {
            }
        }
        //
        for (int ii=0; ii< Free.length; ii++) {
            Fs[ii] = Free[ii][0];
            Ftype[ii] = Free[ii][1];
            try {
                flimits[ii][0] = Double.parseDouble(Free[ii][2]);
                flimits[ii][1] = Double.parseDouble(Free[ii][3]);
                fbits[ii] = Integer.parseInt(Free[ii][4]);
            } catch (NumberFormatException nfe) {
            }
        }
        Vehicle_Type = Types[0][1];
    }
    //
    public static void main (String args[]) {
        //
        JAVA_SPEA masterClass = new JAVA_SPEA();
        masterClass.Arguments = args;
        masterClass.start();
    }
}
```

D.1.2 Output Writing Class

OutputWriter.java

```
//
//  OutputWriter.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Thu Oct 09 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
import java.util.*;
import java.io.*;

public class OutputWriter {
    //
    // A Class whose sole purpose is to create the MSPEA output files.
    //
    String var_names[], filename;
    double variables[][];
    BufferedWriter buffwrite;
    StringBuffer outbuff;

    public OutputWriter(String filename, String[] varname, double[][] vars) {
        this.filename = filename;
        this.var_names = varname;
        this.variables = vars;
```

```
        outbuff = new StringBuffer(5000);
        outbuff.append("Pop Memeber \t");
        for (int ii=0; ii<var_names.length; ii++) {
            outbuff.append(var_names[ii]);
            outbuff.append("\t");
        }
        outbuff.append("\n");
        //

        //
        for (int ii=0; ii< variables.length; ii++) {
            outbuff.append(Integer.toString(ii+1));
            for(int jj=0; jj<variables[ii].length; jj++) {
                outbuff.append("\t");
                outbuff.append(Double.toString(variables[ii][jj]));
            }
            outbuff.append("\n");
        }
        String output = outbuff.toString();
        try {
            buffwrite = new BufferedWriter(new FileWriter(filename, false), 25000);
            buffwrite.write(output, 0, output.length());
            buffwrite.close();
        } catch (IOException e) {
        }
    }

    public OutputWriter(String filename, int[] numbering) {
        this.filename = filename;
        outbuff = new StringBuffer(5000);
        outbuff.append("Pop Memeber\t");

        outbuff.append("Generation\n");
        //

        //
        for (int ii=0; ii< numbering.length; ii++) {
            outbuff.append(Integer.toString(ii+1));
            outbuff.append("\t");
            outbuff.append(Integer.toString(numbering[ii]));
            outbuff.append("\n");
        }
        String output = outbuff.toString();
        try {
            buffwrite = new BufferedWriter(new FileWriter(filename, false), 25000);
            buffwrite.write(output, 0, output.length());
            buffwrite.close();
        } catch (IOException e) {
        }
    }
}
```

D.2 Objective Function Interface

D.2.1 RF_Objective Class

RF_Objective.java.java

```
//
//  RF_Objective.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
import edu.asdl.design.ea.spea.*;
import edu.asdl.design.ea.spea.Objective;
import edu.asdl.design.ea.spea.ParetoIndividual;
import edu.asdl.design.rf.*;
import edu.asdl.design.rf.Vehicle_System;
import edu.asdl.design.rf.Iterator;
import java.lang.reflect.*;
import java.util.*;
//
public class RF_Objective extends Objective implements Cloneable{
    //
    ParetoIndividual Person;
    double Xbounds[][], Tbounds[][], Fbounds[][];
    double Xval[], Yval[], Tval[], Fval[];
    String X[], XX[][], Y[], YY[][], T[], TT[][], type[], Ttype[], F[], FF[][], Ftype[],
        Vehicle_Type;
    int Xgenes[], Fgenes[];
    public Vehicle_System Vehicle;
    public Iterator iterate;
    static final int max_it = 50;
    Object input[][], output[][], Ooo[];
    Class c, cc[], typeclass[], ttypeclass[][];
```

```
Field f[], ff[][];
String fname[], ffname[][], typename[], ttypename[][];
int GW_in, HP_in; // The starting values for the optimization these will always remain the same;
//
//
public RF_Objective() {
  //
}
// Primary Constructor, other constructors are excluded for brevity
public RF_Objective(String vehicletype, String[] Xs, String[] Xtype, String[] Ys, String[] Ts,
                    String[] Fs, String[] Ftype, double[][] Xlimits, double[] Ygoals,
                    double[][] Tlimits, double[][] Flimits, ParetoIndividual Indiv) {
    this.Person = Indiv;
    this.goals = Ygoals;
    this.type = Xtype;
    this.Xbounds = Xlimits;
    this.Tbounds = Tlimits;
    this.Ftype = Ftype;
    this.Fbounds = Flimits;
    this.X = Xs;
    this.Y = Ys;
    this.T = Ts;
    this.F = Fs;
    this.XX = new String[X.length][3];
    this.YY = new String[Y.length][3];
    this.TT = new String[T.length][3];
    this.FF = new String[F.length][3];
    this.Xval = new double[X.length];
    this.results = this.Yval = new double[Y.length];
    this.Tval = new double[T.length];
    this.Fval = new double[F.length];
    this.input = new Object[X.length+F.length][5];
    this.output = new Object[Y.length+T.length][4];
    this.Vehicle_Type = vehicletype;
    initialize_vehicle();
    tokenize();
    iterate = new Iterator(Vehicle, max_it);
}
//
public boolean run() {
    // The portion of the routine that actually places the inputs,
    // runs the RF method and pulls the outputs
    //
    //
    // Set the starting guess;
    //
    c = this.Vehicle.getClass();
    try {
        Field f1 = c.getField("GW_in");
        Field f2 = c.getField("HP_in");
        f1.set(this.Vehicle, new Integer(GW_in));
        f2.set(this.Vehicle, new Integer(HP_in));
    } catch (NoSuchFieldException nsfe) {
        System.out.println("You have broken the code");
    } catch (IllegalAccessException e) {
        System.out.println("You have really broken the code");
    } catch (IllegalArgumentException iae) {
        System.out.println("You have really broken the code " +iae);
    }
    //
    // Set Inputs
    //
    convertGeneToX();
    convertGeneToF();
    buildInput();
    buildOutputStructure();
    placeInputs();
    //
    // Iterate
    //
    iterate.Iterate();
    if (iterate.go == 1) {
        Person.good = false; // Iteration never terminated
    } else {
        Person.good = true; // Iteration terminated successfully
    }
    //
    // Grab outputs
    //
    grabOutput();
    parseOutput();
    return true;
}
//
public double[] getInputs() {
    double Xout[] = new double[Xval.length];
    for (int i=0; i<Xval.length; i++) {
        Xout[i] = Xval[i];
    }
    return Xout;
}
```

```
//
public double[] getFree() {
    double Fout[] = new double[Fval.length];
    // Uses same basic algorithm as getInputs()
    return Fout;
}
//
public double[] getResults() {
    double Yout[] = new double[Yval.length];
    // Uses same basic algorithm as getInputs()
    return Yout;
}
//
public double[] getTracked() {
    double Tout[] = new double[Tval.length];
    // Uses same basic algorithm as getInputs()
    return Tout;
}
//
public double[] getGoals() {
    return goals;
}
//
public void replaceIndividual(ParetoIndividual in) {
    this.Person = in;
}
//
void tokenize() {
    StringTokenizer xt, yt, tt, ft;
    for (int ii=0; ii<X.length; ii++) {
        xt = new StringTokenizer(X[ii],".[]");
        int count=0;
        do {
            try{
                XX[ii][count]=xt.nextToken();
                count++;
            }catch(NoSuchElementException nse) {
            }
        }while(xt.hasMoreElements() == true);
    }
    for (int ii=0; ii<Y.length; ii++) {
        yt = new StringTokenizer(Y[ii],".[]");
        // Use same basic loop as used to capture "xt"
    }
    if (T != null) {
        for (int ii=0; ii<T.length; ii++) {
            tt = new StringTokenizer(T[ii],".[]");
            // Use same basic loop as used to capture "xt"
        }
    }
    if (F != null) {
        for (int ii=0; ii<F.length; ii++) {
            // Use same basic loop as used to capture "xt"
        }
    }
}
//
void buildFields() {
    c = this.Vehicle.getClass();
    f = c.getFields();
    cc = new Class[f.length];
    Ooo = new Object[f.length];
    ff = new Field[f.length][];
    fname = new String[f.length];
    ffname = new String[f.length][];
    typeclass = new Class[f.length];
    ttypeclass = new Class[f.length][];
    typename = new String[f.length];
    ttypename = new String[f.length][];
    for (int ii=0; ii<f.length; ii++) {
        fname[ii] = f[ii].getName();
        typeclass[ii] = f[ii].getType();
        typename[ii] = typeclass[ii].getName();
        if (typename[ii].startsWith("edu.asdl.design.rf.Vehicle")) {
            try {
                Field oo = c.getField(fname[ii]);
                Ooo[ii] = oo.get(this.Vehicle);
                cc[ii] = Ooo.getClass();
                ff[ii] = cc[ii].getFields();
                ffname[ii] = new String[ff.length];
                ttypeclass[ii] = new Class[ff.length];
                ttypename[ii] = new String[ff.length];
                for (int jj=0; jj<ff.length; jj++) {
                    ffname[ii][jj] = ff[ii][jj].getName();
                    ttypeclass[ii][jj] = ff[ii][jj].getType();
                    ttypename[ii][jj] = ttypeclass[ii][jj].getName();
                }
            } catch (NoSuchFieldException nsfe) {
            } catch (IllegalAccessException e) {
            }
```

```
            } else {
                cc[ii] = null;
            }
        }
    }
    //
    public void convertGeneToX() {
        int res[], val[], valcheck[];
        res = Person.bitChrome.getResolution();
        Xgenes = val = Person.realChrome.getGeneValues();
        valcheck = Person.bitChrome.getGeneValues();
        for (int qq=0; qq <val.length; qq++) {
            if (val[qq] != valcheck[qq]) {
                System.out.println("The genevalues don't match: " +qq);
            }
        }
        for (int ii=0; ii<Person.getNumberOptParam(); ii++) {
            double increment = (Xbounds[ii][1] - Xbounds[ii][0]) / (res[ii] - 1.);
            Xval[ii] = val[ii] * increment + Xbounds[ii][0];
        }
    }
    //
    public void convertGeneToF() {
        if (Fval != null) {
            int res[], val[];
            int base = Person.getNumberOptParam();
            res = Person.bitChrome.getResolution();
            Fgenes = val = Person.realChrome.getGeneValues();
            for (int ii=Person.getNumberOptParam(); ii<Person.bitChrome.getGeneNumber(); ii++) {
                double increment = (Fbounds[ii-base][1] - Fbounds[ii-base][0]) / (res[ii] - 1.);
                Fval[ii-base] = val[ii] * increment + Fbounds[ii-base][0];
            }
        }
    }
    //
    public void buildInput() {
        for (int ii=0; ii<Xval.length; ii++) {
            input[ii][0] = new String(XX[ii][0]);
            if (XX[ii][1] != null) {
                input[ii][1] = new String(XX[ii][1]);
            } else {
                input[ii][1] = new String("");
            }
            if (XX[ii][2] != null) {
                input[ii][2] = new String(XX[ii][2]);
            } else {
                input[ii][2] = null;
            }
            input[ii][3] = new String(type[ii]);
            if (type[ii].equalsIgnoreCase("short")) {
                input[ii][4] = new Short((short) Xval[ii]);
            } else if (type[ii].equalsIgnoreCase("int")) {
                input[ii][4] = new Integer((int) Xval[ii]);
            } else if (type[ii].equalsIgnoreCase("float")) {
                input[ii][4] = new Float((float) Xval[ii]);
            }
        }
        if (Fval != null) {
            int base = Xval.length;
            // Use same algorithm for Fs (FF) as was used for Xs (XX)
        }
    }
    //
    public void placeInputs() {
        c = this.Vehicle.getClass();
        for (int ii=0; ii< input.length; ii++) {
            Field f1, f2;
            Object oO;
            Class cC;
            try {
                f1 = c.getField(input[ii][0].toString());
                oO = f1.get(this.Vehicle);
                cC = oO.getClass();
                try {
                    f2 = cC.getField(input[ii][1].toString());
                    int arrayindex = 0;
                    if (input[ii][2] != null){
                        try {
                            arrayindex = Integer.parseInt(input[ii][2].toString());
                        } catch (NumberFormatException nfe) {
                            System.out.println("Array index is not an integer. " +nfe);
                        }
                        try {
                            Object ooO = f2.get(oO);
                            Array.set(ooO, arrayindex, input[ii][4]);
                        } catch (NullPointerException npe) {
                        } catch (IllegalArgumentException iae) {
                        } catch (ArrayIndexOutOfBoundsException aioobe) {
                        }
```

```
                }else {
                    f2.set(oO, input[ii][4]);
                }
            } catch (NoSuchFieldException nsfe) {
                try {
                    f1.set(this.Vehicle, input[ii][4]);
                } catch (IllegalAccessException e) {
                    System.out.println("Error while accessing: " +input[ii][0] +"."
                                       +input[ii][1] +"\n" +e);
                    return;
                }
            }catch (IllegalAccessException e) {
                System.out.println("Error while accessing: " +input[ii][0] +"."
                                   +input[ii][1] +"\n" +e);
                return;
            }
        } catch (NoSuchFieldException nsfe) {
            System.out.println("Variable doesn't exist: " +input[ii][0] +"."
                               +input[ii][1] +"\n" +nsfe);
        } catch (IllegalAccessException e) {
            System.out.println("Error while accessing: " +input[ii][0] +"."
                               +input[ii][1] +"\n" +e);
            return;
        }
    }
}
//
void buildOutputStructure() {
    for (int ii=0; ii<Yval.length; ii++) {
        output[ii][0] = new String(YY[ii][0]);
        if (YY[ii][1] != null) {
            output[ii][1] = new String(YY[ii][1]);
        } else {
            output[ii][1] = new String("");
        }
        if (YY[ii][2] != null) {
            output[ii][2] = new String(YY[ii][2]);
        } else {
            output[ii][2] = null;
        }
        output[ii][3] = new Double(0);
    }
    if (Tval != null) {
        int base = Yval.length;
        // Use same algorithm for Ts (TT) that was used for Ys (YY)
    }
}
//
void grabOutput() {
    c = this.Vehicle.getClass();
    for (int ii=0; ii< output.length; ii++) {
        Field f1, f2;
        Object oO;
        Class cC;
        try {
            f1 = c.getField(output[ii][0].toString());
            oO = f1.get(this.Vehicle);
            cC = oO.getClass();
            try {
                f2 = cC.getField(output[ii][1].toString());
                Object bob = f2.get(oO);
                int arrayindex = 0;
                if (output[ii][2] != null){
                    try {
                        arrayindex = Integer.parseInt(output[ii][2].toString());
                    } catch (NumberFormatException nfe) {
                        System.out.println("Array index is not an integer. " +nfe);
                    }
                    try {
                        output[ii][3] = Array.get(bob, arrayindex);
                    } catch (NullPointerException npe) {
                    } catch (IllegalArgumentException iae) {
                    } catch (ArrayIndexOutOfBoundsException aioobe) {
                    }
                } else {
                    output[ii][3] =  f2.get(oO);
                }
            } catch (NoSuchFieldException nsfe) {
                try {
                    output[ii][3] = f1.get(this.Vehicle);
                } catch (IllegalAccessException e) {
                    System.out.println("Error while accessing: " +output[ii][0] +"."
                                       +output[ii][1] +"\n" +e);
                    return;
                }
            }catch (IllegalAccessException e) {
                System.out.println("Error while accessing: " +output[ii][0] +"."
                                   +output[ii][1] +"\n" +e);
                return;
            } catch (ClassCastException cce) {
```

```
                    System.out.println(cce);
                }
            } catch (NoSuchFieldException nsfe) {
                System.out.println("Variable doesn't exist: " +output[ii][0] +"."
                                   +output[ii][1] +"\n" +nsfe);
            } catch (IllegalAccessException e) {
                System.out.println("Error while accessing: " +output[ii][0] +"."
                                   +output[ii][1] +"\n" +e);
                return;
            }
        }
    }
    //
    void parseOutput() {
        Number intermediate;
        for (int ii=0; ii<Yval.length; ii++) {
            intermediate = (Number)output[ii][3];
            Yval[ii] = intermediate.doubleValue();
        }
        if (Tval != null) {
            int base = Yval.length;
            for (int ii=base; ii<base+Tval.length; ii++) {
                intermediate = (Number)output[ii][3];
                Tval[ii-base] = intermediate.doubleValue();
            }
        }
    }
    //
    public int getIterationNumber() {
        return iterate.iterations;
    }
    //
    public Object clone() {
        // Designed to allow a cloneing of the Objective controller class. Without a clone
        // method all of the RF_Objectives would refer to the same memory location.
        // Clones the individual, code omitted for brevity.
    }
    //
    void initialize_vehicle() {
        Vehicle = new Vehicle_System();
        //
        // Set up the default vehicle inputs. This is necessary to ensure that the RF tool
        // functions when specific values are not varied by the MSPEA Tool
        //
        // Actual default settings removed to save space
        //
        Vehicle.Initiate();
    }
}
```

D.3 Basic Genetic Algorithm

D.3.1 Gene Class

Gene.java

```
package edu.asdl.design.ea.basic;
//
//  Gene.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Fri Sep 19 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
import java.util.Random;
//
public class Gene implements Cloneable  {
    //
    // The foundation Gene class, designed for implementation in any gene based EA/GA.
    // This class implelements a significant number of methods that are used in the more specific
    // implementations of this class
    //
    static Random rngen = new Random();
    byte bit[];
    //
    public Gene(short bitlength) {
        bit = new byte[bitlength];
    }
    //
    short getBitNumber() {
        return (short)bit.length;
    }
    //
    void setBitValue(short bitnumber, byte bitvalue) {
        if ((bitvalue == 1) | (bitvalue == 0)) {
            bit[bitnumber] = bitvalue;
        } else {
```

```
            bit[bitnumber] = 0;
        }
    }
    //
    void setBitHigh(short bitnumber) {
        bit[bitnumber] = 1;
    }

    void setBitLow(short bitnumber) {
        bit[bitnumber] = 0;
    }
    //
    void flipBit(short bitnumber) {
        if (bit[bitnumber] == 0) {
            bit[bitnumber] = 1;
        } else if (bit[bitnumber] == 1) {
            bit[bitnumber] = 1;
        } else {
            bit[bitnumber] = 0;
        }
    }
    //
    byte getBitValue(short bitnumber) {
        return bit[bitnumber];
    }
    //
    int getResolution() {
        int length = bit.length;
        return (int) (Math.pow(2,length));
    }
    //
    int getGeneValue() {
        int length = bit.length;
        int two_power, increment = 0;
        for (int ii=0; ii<length; ii++) {
            two_power = (int)Math.pow(2,ii);
            increment += bit[ii] * two_power;
        }
        return increment;
    }
    //
    public void initRandom() {

        int val;
        for (int ii=0; ii<bit.length; ii++) {
            val = rngen.nextInt(2);
            bit[ii] = (byte)val;
        }
    }
    //
    public Object clone() throws CloneNotSupportedException {
    } // Clone code omitted for brevity
}
```

D.3.2 Base Chromosome Class

Chromosome.java

```
package edu.asdl.design.ea.basic;
//
//  Chromosome.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Fri Sep 19 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//

import java.util.Random;

public class Chromosome implements Cloneable {
    //
    // The foundation Chromosome class, designed for implementation in any gene based EA/GA.
    // This class implelements a significant number of methods that are used in the more specific
    // implementations of this class
    //
    public Gene gene[];
    int resolution[], values[];
    public short number_of_opt_param;
    //
    public Chromosome(short numbergenes, short[] numberbits) {
        gene = new Gene[numbergenes];
        values = new int[numbergenes];
        short length = (short)numberbits.length;
        for (int ii=0; ii< length; ii++) {
            gene[ii] = new Gene(numberbits[ii]);
        }
    }
    //
    public void setNumberOptParam(short set) {} // Code omitted for brevity
    //
    public short getNumberOptParam() {} // Code omitted for brevity
```

```
//
public short getGeneNumber() {} // Code omitted for brevity
//
public short getBitNumber(short gene_num) {} // Code omitted for brevity
//
public int[] getResolution() {
    int length, resolution[];
    length = gene.length;
    resolution = new int[length];
    for (int ii=0; ii<length; ii++) {
        resolution[ii] = gene[ii].getResolution();
    }
    return resolution;
}
//
public int[] getGeneValues() {
    int length;
    length = gene.length;
    values = new int[length];
    for (int ii=0; ii<length; ii++) {
        values[ii] = gene[ii].getGeneValue();
    }
    return values;
}
//
public void setBitValue(short genenumber, short bitnumber, byte bitvalue) {
    gene[genenumber].setBitValue(bitnumber, bitvalue);
} // Code omitted for brevity
//
public void setBitHigh(short genenumber, short bitnumber) { } // Code omitted for brevity

public void setBitLow(short genenumber, short bitnumber) {} // Code omitted for brevity

public void flipBit(short genenumber, short bitnumber) {} // Code omitted for brevity

public byte getBitValue(short genenumber, short bitnumber) {} // Code omitted for brevity
//
public Gene tradeGene(short genenumber, Gene tradeINgene) {
    Gene tradeOUTgene;
    tradeOUTgene = gene[genenumber];
    gene[genenumber] = tradeINgene;
    return tradeOUTgene;
}
//
public void initRandom() {
    for (int ii=0; ii<gene.length; ii++) {
        gene[ii].initRandom();
    }
}
//
public Gene cloneGene(short genenumber) throws CloneNotSupportedException {
} // Clone code omitted for brevity
//
public Object clone() throws CloneNotSupportedException {
} // Clone code omitted for brevity
//
public Gene recombineGenes(short genenumber, Gene tradegene2) {
    Gene gene1, gene2, tradegene1;
    short numbits1, numbits2;
    byte bitstore[][];
    tradegene1 = this.gene[genenumber];
    numbits1 = (short)tradegene1.bit.length;
    numbits2 = (short)tradegene2.bit.length;
    gene1 = new Gene(numbits1);
    gene2 = new Gene(numbits2);
    Random rand = new Random();
    int val;
    if (numbits1 != numbits2) {

    }
    bitstore = new byte[numbits1][2];
    for (short ii=0; ii<numbits1; ii++) {
        bitstore[ii][0] = tradegene1.getBitValue(ii);
        bitstore[ii][1] = tradegene2.getBitValue(ii);
    }
    for (short jj=0; jj<numbits1; jj++) {
        val = rand.nextInt(2);
        if (val == 1) {
            gene1.setBitValue(jj, bitstore[jj][1]);
            gene2.setBitValue(jj, bitstore[jj][0]);
        } else {
            gene1.setBitValue(jj, bitstore[jj][0]);
            gene2.setBitValue(jj, bitstore[jj][1]);
        }
    }
    tradegene1 = gene1;
    return gene2;
}
//
public boolean equals(Chromosome chrome) {
    boolean equal = false;
    for (int ii=0; ii<gene.length; ii++) {
```

```
            if (gene[ii].getGeneValue() != chrome.gene[ii].getGeneValue()) {
                return false;
            }
        }
        return true;
    }
}
```

D.3.3 Real Value Chromosome Class

RealValChromosome.java

```
package edu.asdl.design.ea.basic;

//
//  RealValChromosome.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Wed Sep 24 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//

public class RealValChromosome implements Cloneable{
    //
    // A sub-class of Chromosome that handels the calculation of the real values of a chromosome
    // string. The primary purpose of this is to abstract the functionality from the general
    // chromosome class which may not need it.
    //
    int realGene[];
    private Chromosome chrome;
    public short number_of_opt_param;
    //
    public RealValChromosome(Chromosome chromein) {
        this.chrome = chromein;
        short num_genes = chrome.getGeneNumber();
        realGene = new int[num_genes];
        realGene = chrome.getGeneValues();
        this.number_of_opt_param = chrome.number_of_opt_param;
    }
    //
    public int getSingleValue(int genenumber) {} // Code omitted for Brevity
    //
    public int[] getGeneValues() {} // Code omitted for Brevity
    //
    public void update() {
        realGene = chrome.getGeneValues();
    }
    //
    public void update(Chromosome in) {
        this.chrome = in;
        realGene = chrome.getGeneValues();
    }
    //
    public boolean equals(RealValChromosome chrome) {
        boolean equal = false;
        int gene1, gene2;
        for (int ii=0; ii<realGene.length; ii++) {
            gene1 = realGene[ii];
            gene2 = chrome.realGene[ii];
            if (gene1 != gene2) {
                return false;
            }
        }
        return true;
    }
    //
    public Object clone() throws CloneNotSupportedException {} // Clone code omitted for brevity
}
```

D.3.4 Individual Abstract Class

Individual.java

```
package edu.asdl.design.ea.basic;
//
//  Individual.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Fri Sep 19 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//

public abstract class Individual implements Cloneable{
    //
    // The abstract Individual class, designed for implementation in any  EA/GA.
    // This class implelements a significant number of methods that are used in the more specific
    // implementations of this class
    //
    public Chromosome bitChrome;
```

```
    public RealValChromosome realChrome;
    public Fitness indivFit;
    public double fitness_value;
    protected short numgenes, numbits[];

    abstract void update();

    abstract boolean mutate(int probability);

    abstract void corrolateChromosomes();

    public abstract Object clone() throws CloneNotSupportedException;
}
```

D.3.5 Simple Individual Class

SimpleIndividual.java

```
package edu.asdl.design.ea.basic;

//
//  SimpleIndividual.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Wed Sep 24 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//

import java.util.Random;

public class SimpleIndividual extends Individual implements Cloneable {
    //
    // A more specific extension of the Idividual Class. It actually implements many Methods
    //
    public SimpleIndividual(short numgenes, short[] numbits) {
        bitChrome = new Chromosome(numgenes, numbits);
        bitChrome.initRandom();
        realChrome = new RealValChromosome(bitChrome);
        this.numgenes = numgenes;
        this.numbits = numbits;
    }

    void corrolateChromosomes() {
        realChrome.update();
    }

    public void update() {
        corrolateChromosomes();
    }

    public boolean mutate(int probability) {
        //
        // Simple bit flip mutation
        //
        boolean bool= false;
        float prob = probability * 1e-2f;
        Random rand = new Random();
        short ind1, ind2;
        float rand_val = rand.nextFloat();
        if (rand_val <= prob) {
            ind1 = (short) rand.nextInt(numgenes);
            ind2 = (short)rand.nextInt(numbits[ind1]);
            bool = true;
            bitChrome.flipBit(ind1, ind2);
        }
        return bool;
    }

    public boolean crossover(int probability, SimpleIndividual tradeindv) {
        //
        //Simple Gene Swap Crossover
        //
        boolean bool= false;
        float prob = probability * 1e-2f;
        Random rand = new Random();
        Gene trade_gene;
        Chromosome trade_chrom = tradeindv.bitChrome;

        float rand_val = rand.nextFloat();
        if (rand_val <= prob) {
            short index = (short)rand.nextInt(bitChrome.getGeneNumber());
            trade_gene = trade_chrom.gene[index];
            trade_gene = bitChrome.tradeGene(index, trade_gene);
            bool = true;
        }
        return bool;
    }

    public boolean equals(SimpleIndividual simpind) {
        //
        // Checks if the two indivduals are Genetically Identical\
        //
        boolean bool = true;
```

```
        Gene this_gene[], that_gene[];
        short this_gene_length, that_gene_length, this_bits, that_bits;
        this_gene_length = bitChrome.getGeneNumber();
        that_gene_length = simpind.bitChrome.getGeneNumber();
        if (this_gene_length != that_gene_length) {
            bool = false;
            return bool;
        }
        for (short ii=0; ii<this_gene_length; ii++) {
            this_bits = bitChrome.gene[ii].getBitNumber();
            that_bits = simpind.bitChrome.gene[ii].getBitNumber();
            if (this_bits != that_bits) {
                bool = false;
                return bool;
            }
            for (short jj=0; jj<this_bits; jj++) {
                if (bitChrome.gene[ii].getBitValue(jj)
                    != simpind.bitChrome.gene[ii].getBitValue(jj)) {
                    bool = false;
                    return bool;
                }
            }
        }
        return bool;
    }

    public Fitness getFitness() {
        return indivFit;
    }

    public void putFitness(Fitness fitin) {
        indivFit = fitin;
    }

    public Object clone() throws CloneNotSupportedException {
        SimpleIndividual clonedIndiv = new SimpleIndividual(numgenes, numbits);
        return clonedIndiv;
    }
}
```

D.3.6 Population Abstract Class

Population.java

```
package edu.asdl.design.ea.basic;
//
//  Population.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Fri Sep 19 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//

public abstract class Population {
    //
    // The abstract Population class, designed for implementation in any  EA/GA.
    // This class implelements a significant number of methods that are used in the more specific
    // implementations of this class
    //
    protected Individual individuals[];

    public abstract Individual getIndividual(int indvnum);
    public short numgenes, numbits[];

    public int getNumberIndividuals() {
        return individuals.length;
    }

    public short getNumGenes() {
        return numgenes;
    }

    public short[] getNumBits() {
        return numbits;
    }
}
```

D.3.7 Simple Population Class

SimplePopulation.java

```
package edu.asdl.design.ea.basic;
//
//  SimplePopulation.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Thu Oct 02 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
```

```
public class SimplePopulation extends Population implements Cloneable{
    //
    // A more specific extension of the Idividual Class. It is the basis for the more specific
    // population classes, few of the methods are more than shells.
    //
    public SimplePopulation(int num_of_individuals, short numgenes, short[] numbits) {
        //super(num_of_individuals, numgenes, numbits);
    }

    public Object clone() throws CloneNotSupportedException {
        return this;
    }

    public Individual getIndividual(int indvnum) {
        return individuals[indvnum];
    }
}
```

D.3.8 Fitness Abstract Class

Fitness.java

```
package edu.asdl.design.ea.basic;
//
//  Fitness.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Fri Sep 19 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//

public abstract class Fitness {
    //
    // The abstract Fitness class, designed for implementation in any  EA/GA.
    // This class implelements a significant number of methods that are used in the more specific
    // implementations of this class
    //
    protected float weighting[];
    public int objectives;
    protected double fitness_value;
    protected double fitness_indv[];

    public Fitness(){
        objectives = 1;
        weighting = new float[1];
        fitness_indv = new double[1];
        weighting[0] = 1.0f;
    }

    public Fitness(int number_fit_objects) {
        this.objectives = number_fit_objects;
        weighting = new float[number_fit_objects];
        fitness_indv = new double[number_fit_objects];
        float weight = 1f/number_fit_objects;
        for (int ii=0; ii<number_fit_objects; ii++) {
            weighting[ii] = weight;
        }
    }

    public Fitness(int number_fit_objects, float[] weights){
        this.objectives = number_fit_objects;
        fitness_indv = new double[number_fit_objects];
        if (number_fit_objects != weights.length) {

        }
        this.weighting = weights;
    }

    public double getFitValue() {
        return fitness_value;
    }

    public void putFitValue(double value) {
        fitness_value = value;
    }

    public double getIndivFitValue(int s) {
        return fitness_indv[s];
    }

    public void putIndivFitValue(int s, double value) {
        fitness_indv[s] = value;
    }

    public double[] getTracked() {
        double stan[] = new double[1];
        stan[0] = 0;
        return stan;
    }

    public double[] getTrackedNormal() {
        double stan[] = new double[1];
        stan[0] = 0;
```

```
        return stan;
    }

    abstract public Object clone() throws CloneNotSupportedException;

    public abstract boolean runObjective();

    public void calcTotalFitValue() {
        fitness_value = 0;
        for (int ii=0; ii< objectives; ii++) {
            fitness_value += fitness_indv[ii];
        }
    }
}
```

D.3.9 Tournament Class

Tournement.java

```
package edu.asdl.design.ea.basic;
//
//  Tournement.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Wed Oct 01 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
import java.util.Random;

public class Tournement {
    //
    // Basic Tournment Selection method class, for uses in general EA/GAs
    //
    public Population internalPop, childPop;
    public int population_number, internal_length;
    public short  numgenes, numbits[];

    public Tournement() {
    }

    public Tournement(SimplePopulation pop) {
        internalPop = pop;
        internal_length = population_number = internalPop.getNumberIndividuals();
        numgenes = internalPop.getNumGenes();
        numbits = internalPop.getNumBits();
        createChildren();
    }

    public Tournement(SimplePopulation pop, int number_of_children) {
        internalPop = pop;
        internal_length = internalPop.getNumberIndividuals();
        population_number = number_of_children;
        numgenes = internalPop.getNumGenes();
        numbits = internalPop.getNumBits();
        createChildren();
    }

    void createChildren() {
        childPop = new SimplePopulation(population_number, numgenes, numbits);
    }

    public boolean compete() {
        boolean success = false;
        Random randgen = new Random();
        int popnumber[][] = new int[population_number][2];
        for (int ii=0; ii<population_number; ii++) {
            popnumber[ii][0] = randgen.nextInt(internal_length+1);
            popnumber[ii][1] = randgen.nextInt(internal_length+1);
        }
        //
        double fit_1, fit_2;
        int ind_1, ind_2;
        for (int ii=0; ii < population_number; ii++) {
            fit_1 = internalPop.individuals[popnumber[ii][0]].indivFit.getFitValue();
            fit_2 = internalPop.individuals[popnumber[ii][1]].indivFit.getFitValue();
            ind_1 = popnumber[ii][0];
            ind_2 = popnumber[ii][1];
            if (fit_1 >= fit_2) {
                try {
                   childPop.individuals[ii] = (Individual)internalPop.individuals[ind_1].clone();
                } catch (CloneNotSupportedException cnse) {
                }
            } else {
                try {
                    childPop.individuals[ii] = (Individual)internalPop.individuals[ind_2].clone();
                } catch (CloneNotSupportedException cnse) {
                }
            }
        }
        return success;
    }

    public Population getChildPop() {
```

```
        return childPop;
    }
}
```

D.4 Modified Strength Pareto Evolutionary Algorithm

D.4.1 Pareto Individual Class

ParetoIndividual.java

```
package edu.asdl.design.ea.spea;
//
//  ParetoIndividual.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Wed Sep 24 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
import edu.asdl.design.ea.basic.*;
import edu.asdl.design.ea.basic.Population;
import edu.asdl.design.ea.basic.SimpleIndividual;
//
public class ParetoIndividual extends SimpleIndividual implements Cloneable {
    //
    // The basic individual for the MSPEA algorithm, all of the important work is performed here.
    //
    protected boolean isDominated, isDominatedX, isDominatedY, isDominatedT, fitnessEval=false;
    public float xStrength, yStrength, tStrength, strength;
    public short number_of_opt_param;
    protected XParetoFitness indivFit;
    public boolean good = true;
    //
    public ParetoIndividual(short numgenes, short[] numbits, Objective object) {
        super(numgenes, numbits);
        this.indivFit = new XParetoFitness(object);
    }
    //
    public ParetoIndividual(short numgenes, short[] numbits) {} // Code omitted for brevity
    //
    public void setNumberOptParam(short set) {
        number_of_opt_param = set;
        bitChrome.setNumberOptParam(set);
    }
    //
    public void initialChromosome() {} // Code omitted for brevity
    //
    public void createFitness(Objective object) {} // Code omitted for brevity
    //
    public void setObjective(Objective object) {} // Code omitted for brevity
    //
    public short getNumberOptParam() {} // Code omitted for brevity
    //
    public void setIsDominated() {} // Code omitted for brevity
    //
    public void setIsNotDominated() {} // Code omitted for brevity
    //
    public boolean IsDominated() {} // Code omitted for brevity
    //
    public boolean IsDominatedX() {} // Code omitted for brevity
    //
    public boolean IsDominatedY() {} // Code omitted for brevity
    //
    public boolean IsDominatedT() {} // Code omitted for brevity
    //
    public void setDominationY(boolean in) {} // Code omitted for brevity
    //
    public void setDominationX(boolean in) {} // Code omitted for brevity
    //
    public void setDominationT(boolean in) {} // Code omitted for brevity
    //
    public void alignDomination() {
        isDominated = isDominatedX || isDominatedY || isDominatedT;
    }
    //
    public double getIndivFitValue(int s) {} // Code omitted for brevity
    //
    public double[] getTracked() { } // Code omitted for brevity
    //
    public boolean dominates(ParetoIndividual indiv) {
        //
        // Determines if this individual "dominatates" the ParetoIndividual indiv in the Response
        // space.
        //
```

```
        boolean dominates = false;
        short numberobjectives = (short) indivFit.objectives;
        double objec_1, objec_2;
        for (int ii = 0 ; ii < numberobjectives; ii++) {
            objec_1 = this.getIndivFitValue(ii);
            objec_2 = indiv.getIndivFitValue(ii);
            if (objec_2 > objec_1) {
                return false;
            } else if (objec_2 < objec_1) {
                dominates = true;
            }
        }
        return dominates;
    }
    //
    public boolean dominatesX(Individual indiv, int[] nominal) {
        //
        // Determines if this individual "dominatates" the ParetoIndividual indiv in the Parameter
        // space.
        //
        boolean dominates = false;
        int numberXs = nominal.length;
        int x_1[], x_2[], x_nom[]= nominal ;
        float r_1, r_2, abs_1, abs_2;
        x_1 = bitChrome.getGeneValues();
        x_2 = indiv.bitChrome.getGeneValues();
        for (int ii=0; ii< numberXs; ii++) {
            r_1 = (float)(x_1[ii] - x_nom[ii]) / x_nom[ii];
            r_2 = (float)(x_2[ii] - x_nom[ii]) / x_nom[ii];
            abs_1 = Math.abs(r_1);
            abs_2 = Math.abs(r_2);
            if ((r_1/abs_1 - r_2/abs_2) == 0f) {
                if (abs_1 < abs_2) {
                    return false;
                } else if (abs_1 > abs_2) {
                    dominates = true;
                }
            }else if ((r_1/abs_1 - r_2/abs_2) != 0f) {
                return false;
            }
        }
        return dominates;
    }
    //
    public boolean dominatesT(ParetoIndividual indiv, double[] nominal) {
        //
        // Determines if this individual "dominatates" the ParetoIndividual indiv in the Parameter
        // space, which is part of the Objective output. Method is substantially similar to
        // dominatesX.
        //
    }
    //
    public boolean covers(ParetoIndividual indiv) {
        //
        // Determines if this individual "covers" the ParetoIndividual indiv in the Response
        // space.
        //
        short numberobjectives = (short) indivFit.objectives;
        double objec_1, objec_2;
        for (int ii = 0 ; ii < numberobjectives; ii++) {
            objec_1 = indivFit.getIndivFitValue(ii);
            objec_2 = indiv.indivFit.getIndivFitValue(ii);
            if (objec_2 > objec_1) {
                return false;
            }
        }
        return true;
    }
    //
    public boolean coversX(ParetoIndividual indiv, int[] nominal) {
        //
        // Determines if this individual "covers" the ParetoIndividual indiv in the Parameter
        // space.
        //
        int numberXs = nominal.length;
        int x_1[], x_2[], x_nom[]= nominal ;
        float r_1, r_2, abs_1, abs_2;
        x_1 = bitChrome.getGeneValues();
        x_2 = indiv.bitChrome.getGeneValues();
        for (int ii=0; ii< numberXs; ii++) {
            r_1 = (float)(x_1[ii] - x_nom[ii]) / x_nom[ii];
            r_2 = (float)(x_2[ii] - x_nom[ii]) / x_nom[ii];
            abs_1 = Math.abs(r_1);
            abs_2 = Math.abs(r_2);
            if ((r_1/abs_1 - r_2/abs_2) == 0f) {
                if (abs_1 < abs_2) {
                    return false;
                }
            }else if ((r_1/abs_1 - r_2/abs_2) != 0f) {
                return false;
            }
        }
```

```
        return true;
    }
    //
    public boolean coversT(ParetoIndividual indiv, double[] nominal) {
        //
        // Determines if this individual "covers" the ParetoIndividual indiv in the "Tracked"
        // space. Method is substantially similar to coversX.
        //
    }
    //
    public boolean equals(ParetoIndividual indiv) {
        //
        // Determines if two indivduals are equal to each other in "Response" space
        //
        short numberobjectives = (short) indivFit.objectives;
        double objec_1, objec_2;
        for (int ii = 0 ; ii < numberobjectives; ii++) {
            objec_1 = indivFit.getIndivFitValue(ii);
            objec_2 = indiv.indivFit.getIndivFitValue(ii);
            if (objec_2 != objec_1) {
                return false;
            }
        }
        return true;
    }
    //
    public boolean equalsX(ParetoIndividual indiv, int[] nominal) {
        //
        // Determines if two indivduals are equal to each other in "Parameter" space
        //
        int numberXs = nominal.length;
        int x_1[], x_2[], x_nom[]= nominal ;
        float r_1, r_2, abs_1, abs_2;
        x_1 = bitChrome.getGeneValues();
        x_2 = indiv.bitChrome.getGeneValues();
        for (int ii=0; ii< numberXs; ii++) {
            r_1 = (float)(x_1[ii] - x_nom[ii]) / x_nom[ii];
            r_2 = (float)(x_2[ii] - x_nom[ii]) / x_nom[ii];
            abs_1 = Math.abs(r_1);
            abs_2 = Math.abs(r_2);
            if ((r_1/abs_1 - r_2/abs_2) == 0f) {
                if (abs_1 != abs_2) {
                    return false;
                }
            }else if ((r_1/abs_1 - r_2/abs_2) != 0f) {
                return false;
            }
        }
        return true;
    }
    //
    public boolean equalsT(ParetoIndividual indiv, double[] nominal) {
        //
        // Determines if two indivduals are equal to each other in "Tracked" space. Method is
        // substantially similar to equalssX.
        //
    }
    //
    public double XDistance(ParetoIndividual indiv) {
        //
        // Calculates the distance between this and individual "indiv" in the
        // parameter space.
        //
        int resolution[] = this.bitChrome.getResolution();
        int value_1[] = this.bitChrome.getGeneValues();
        int value_2[] = indiv.bitChrome.getGeneValues();
        double n_value_1[], n_value_2[];
        //int res_length = resolution.length;
        int res_length = this.bitChrome.getNumberOptParam();
        double distance=0.;
        n_value_1 = new double[res_length];
        n_value_2 = new double[res_length];
        for (int ii=0; ii<res_length; ii++) {
            n_value_1[ii] = value_1[ii]/(double)resolution[ii];
            n_value_2[ii] = value_2[ii]/(double)resolution[ii];
            distance += Math.pow((n_value_1[ii] - n_value_2[ii]), 2);
        }
        distance = Math.sqrt(distance);
        return distance;
    }
    //
    public double TDistance(ParetoIndividual indiv) {
        //
        // Calculates the distance between this and individual "indiv" in the
        // parameter space. Method is substantially similar to XDistance.
        //
    }
    //
    public Object clone() throws CloneNotSupportedException {
        // Clones the individual, code omitted for brevity
    }
```

```
    //
    public float Strength() {
        // Dtermines combined strength
        return strength = xStrength + yStrength + tStrength;
    }
}
```

D.4.2 Pareto Population Class

ParetoPopulation.java

```
package edu.asdl.design.ea.spea;
//
//  ParetoPopulation.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Wed Sep 24 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//

import edu.asdl.design.ea.basic.*;
import edu.asdl.design.ea.basic.Population;

public class ParetoPopulation extends Population implements Cloneable{
    //
    // The basic population for the MSPEA algorithm, all of the important work is performed here.
    //
    protected ParetoIndividual individuals[];
    protected int nominal[], number_of_indiv, external_indiv;
    protected int ExternalPopNumber;
    protected Objective objective;
    public short number_of_opt_param;
    public short num_tracked;
    //
    public ParetoPopulation() {}
    // Primary Constructor only, included for brevity
    public ParetoPopulation(int num_of_individuals, short numgenes, short[] numbits,
                            Objective fitin) {
        this.objective = fitin;
        individuals = new ParetoIndividual[num_of_individuals];
        number_of_indiv = num_of_individuals;
        external_indiv = number_of_indiv >> 1;
        for (int ii=0; ii<num_of_individuals; ii++) {
            individuals[ii] = new ParetoIndividual(numgenes, numbits, fitin);
        }
    }
    //
    // Paretopopulation methods for obtaining and setting properties omitted for brevity
    //
    public Individual getIndividual(int indvnum) {
        return individuals[indvnum];
    }
    //
    public void putIndividual(int indvnum, ParetoIndividual indiv) {
        individuals[indvnum] = indiv;
    }
    //
    public void setObjective(int indvnum, Objective object) {
        individuals[indvnum].setObjective(object);
    }
    //
    public double[][] getXs() {
        double out[][] = new double[countMembers()][];
        for (int ii=0; ii<countMembers(); ii++) {
            out[ii] = individuals[ii].indivFit.getInputs();
        }
        return out;
    }
    //
    public double[][] getYs() {} // Code omitted for brevity, substanitally similar to getXs()
    //
    public double[][] getTs() {} // Code omitted for brevity, substanitally similar to getXs()
    //
    public double[][] getFs() {} // Code omitted for brevity, substanitally similar to getXs()
    //
    public boolean[] getDominationStatus() {
        boolean out[] = new boolean[individuals.length];
        for (int ii=0; ii< individuals.length; ii++) {
            individuals[ii].alignDomination();
            out[ii] = individuals[ii].IsDominated();
        }
        return out;
    }
    //
    public boolean[] getXDominationStatus() {
        boolean out[] = new boolean[individuals.length];
        for (int ii=0; ii< individuals.length; ii++) {
            out[ii] = individuals[ii].IsDominatedX();
        }
        return out;
```

```
    }
    //
    public boolean[] getYDominationStatus() {
    } // Code omitted for brevity, substanitally similar to getXDominationStatus()
    //
    public boolean[] getTDominationStatus() {
    } // Code omitted for brevity, substanitally similar to getXDominationStatus()
    //
    public boolean dominatesY(int indvnum_1, int indvnum_2) {
        //
        // returns true if invidual[indvnum_1] dominates indvnum_2
        //
        boolean dominates=false;
        dominates = individuals[indvnum_1].dominates(individuals[indvnum_2]);
        if (dominates) {
            individuals[indvnum_2].setDominationY(dominates);
        }
        return dominates;
    }
    //
    public boolean dominatesX(int indvnum_1, int indvnum_2, int[] nominal) {
        //
        // returns true if invidual[indvnum_1] dominates indvnum_2
        //
        boolean dominates=false;
        dominates = individuals[indvnum_1].dominatesX(individuals[indvnum_2], nominal);
        if (dominates) {
            individuals[indvnum_2].setDominationX(dominates);
        }
        return dominates;
    }
    //
    public boolean dominatesT(int indvnum_1, int indvnum_2, double[] nominal) {
    } // Code omitted for brevity, substanitally similar to dominatesX()
    //
    public boolean dominates(int indvnum_1, int indvnum_2, int[] nominal, double[] nomtrack) {
        //
        // returns true if invidual[indvnum_1] dominates indvnum_2
        //
        boolean dominatesx, dominatesy, dominatest;
        dominatesx = dominatesX(indvnum_1,indvnum_2 , nominal);
        dominatesy = dominatesY(indvnum_1,indvnum_2);
        dominatest = dominatesT(indvnum_1,indvnum_2 , nomtrack);
        return dominatesx & dominatesy & dominatest;
    }
    //
    public float YStrength(ParetoIndividual indiv_1) {
        //
        // This routine calculates the Y strength fitness
        //
        float bob=0; // = new float[10];
        int N = number_of_indiv, numerator=0;
        float denom = N + 1f;
        boolean covered = false;
        if (indiv_1.isDominatedY) {
            //
            // For points dominated in the Y Space
            //
            for (int ii=0; ii<individuals.length; ii++) {
                if (!individuals[ii].isDominatedY) {
                    covered = individuals[ii].covers(indiv_1);
                    if (covered) {
                        numerator++;
                    }
                } else {
                    covered = false;
                }
            }
            numerator = numerator + N + 1;
        } else if (!indiv_1.isDominatedY) {
            //
            // For points nondominated in the Y Space
            //
            for (int ii=0; ii<individuals.length; ii++) {
                if (individuals[ii].isDominatedY) {
                    covered = indiv_1.covers(individuals[ii]);
                    if (covered) {
                        numerator++;
                    }
                } else {
                    covered = false;
                }
            }
        }
        bob = numerator / denom;
        indiv_1.yStrength = bob;
        return bob;
    }
    //
    public float XStrength(ParetoIndividual indiv_1, int[] nominal) {
        //
        // This routine calculates the X strength fitness
        //
```

```
        float bob=0; // = new float[10];
        int N = number_of_indiv, numerator=0;
        float denom = N + 1f;
        boolean covered = false;
        if (indiv_1.isDominatedX) {
            //
            // For points dominated in the X Space
            //
            for (int ii=0; ii<individuals.length; ii++) {
                if (!individuals[ii].isDominatedX) {
                    covered = individuals[ii].coversX(indiv_1, nominal);
                    if (covered) {
                        numerator++;
                    }
                } else {
                    covered = false;
                }
            }
            numerator = numerator + N + 1;
        } else if (!indiv_1.isDominatedX) {
            //
            // For points nondominated in the X Space
            //
            for (int ii=0; ii<individuals.length; ii++) {
                if (individuals[ii].isDominatedX) {
                    covered = indiv_1.coversX(individuals[ii], nominal);
                    if (covered) {
                        numerator++;
                    }
                } else {
                    covered = false;
                }
            }
        }
        bob = numerator / denom;
        indiv_1.xStrength = bob;
        return bob;
    }
    //
    public float TStrength(ParetoIndividual indiv_1, double[] nominal) {
        //
        // This routine calculates the T strength fitness, and is substatially similar to
        // XStrength()
        //
    }
}
```

D.4.3 External Pareto Population Class

ExternalParetoPopulation.java

```
package edu.asdl.design.ea.spea;
//
//  ExternalParetoPopulation.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Fri Sep 26 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
public class ExternalParetoPopulation extends ParetoPopulation implements Cloneable{
    //
    // The external population for the MSPEA algorithm. This is where the non-dominated points
    // are stored and later output.
    //
    ParetoPopulation Internal;
    int extra_individuals=0;
    ParetoIndividual extraIndivual[];
    final double max_double = Double.MAX_VALUE;
    int gen_num_entered[], gen_num_ente_ext[];
    //
    public ExternalParetoPopulation(int num_of_individuals, ParetoPopulation internal ) {
        individuals = new ParetoIndividual[num_of_individuals];
        extraIndivual = new ParetoIndividual[num_of_individuals];
        gen_num_entered = new int[num_of_individuals];
        gen_num_ente_ext = new int[num_of_individuals];
        this.Internal = internal;
        this.number_of_indiv = num_of_individuals;
    }
    //
    // Methods to obtain or set ExternalParetoPopulation properties are omitted for brevity
    //
    public double[][] getXs() {
        // Substantially similar to the method of the same name in the ParetoPopulation class
    }
    //
    public double[][] getYs() {
        // Substantially similar to the method of the same name in the ParetoPopulation class
    }
    //
    public double[][] getTs() {
        // Substantially similar to the method of the same name in the ParetoPopulation class
```

```
    }
    //
    public double[][] getFs() {
        // Substantially similar to the method of the same name in the ParetoPopulation class
    }
    //
    public int[] getGenerationNumbers() {
        int out[] = new int[countMembers()];
        for (int ii=0; ii<countMembers(); ii++) {
            out[ii] = gen_num_entered[ii];
        }
        return out;
    }
    //
    public boolean copyTo(ParetoIndividual Individual, int[] nominal, double[] nominalTracked,
                          int gen_num) {
        boolean worked = false;
        int ii;
        for (ii=0; ii<number_of_indiv; ii++) {
            if (individuals[ii] == null) {
                try {
                    individuals[ii] = (ParetoIndividual)Individual.clone();
                    gen_num_entered[ii] = gen_num;
                } catch (CloneNotSupportedException cnse) {
                    return false;
                }
                return true;
            }
            if (individuals[ii].bitChrome.equals(Individual.bitChrome)) {
                return false;
            }
            if (individuals[ii].realChrome.equals(Individual.realChrome)) {
                return false;
            }
            boolean dominatesX = Individual.dominatesX(individuals[ii], nominal);
            boolean dominatesY = Individual.dominates(individuals[ii]);
            boolean dominatesT = Individual.dominatesT(individuals[ii], nominalTracked);
            boolean dominatedX = individuals[ii].dominatesX(Individual, nominal);
            boolean dominatedY = individuals[ii].dominates(Individual);
            boolean dominatedT = individuals[ii].dominatesT(Individual, nominalTracked);
            if ( dominatedX || dominatedY || dominatedT ) {
                return false;
            }
            if (dominatesX & dominatesY & dominatesT) {
                try {
                    individuals[ii] = (ParetoIndividual)Individual.clone();
                    gen_num_entered[ii] = gen_num;
                } catch (CloneNotSupportedException cnse) {
                    return false;
                }

                return true;
            }
        }
        if (ii==number_of_indiv-1) {
            try {
                extraIndivual[extra_individuals] = (ParetoIndividual)Individual.clone();
                gen_num_ente_ext[ii] = gen_num;
            }catch (CloneNotSupportedException cnse) {
                System.out.println(cnse);
            }
            extra_individuals++;
            return true;
        }
        return worked;
    }
    //
    boolean cluster() {
        int total_size = extra_individuals + individuals.length, count;
        // Create Temporary Indivdual Vector for Clustering;
        ParetoIndividual tempIndividual[] = new ParetoIndividual[total_size];
        int temp_gen_num[] = new int[total_size];
        // Fill temporary vector
        for (count=0; count<individuals.length; count++) {
            tempIndividual[count] = individuals[count];
            temp_gen_num[count] = gen_num_entered[count];
        }
        for (int ii=count; ii < total_size; ii++) {
            tempIndividual[ii] = extraIndivual[ii-count];
            temp_gen_num[ii] = gen_num_ente_ext[ii-count];
        }
        // Begin Clustering
        //
        // This routine taken from Population.cpp
        // By: Eckart Zitzler & Paul E. Sevinc
        // Adapted using X instead of Y distances
        //
        int nr = total_size,
            nrOfClusters = nr;
        int clusterBeg[] = new int[nr];
        int indLinks[] = new int[nr];
        //
```

```
        for (int ii=0; ii<nr; ii++) {
            clusterBeg[ii] = ii;
            indLinks[ii] = nr;
        }
        //
        while (nrOfClusters > number_of_indiv) {
            double minDis = max_double;
            int cluster1=0, cluster2=0;
            //
            for (int ii=0; ii< nrOfClusters; ii++ ) {
                for (int jj=ii+1; jj< nrOfClusters; jj++) {
                    double curDis = 0.;
                    int pairCount = 0;
                    int x = clusterBeg[ii];
                    //
                    while (x != nr) {
                        int y = clusterBeg[jj];
                        ParetoIndividual indiv = tempIndividual[x];
                        //
                        while (y != nr) {
                            curDis += indiv.XDistance(tempIndividual[y]);
                            curDis += indiv.TDistance(tempIndividual[y]);
                            ++pairCount;
                            y = indLinks[y];
                        }
                        x = indLinks[x];
                    }
                    curDis /= pairCount;
                    if (curDis < minDis) {
                        cluster1 = ii;
                        cluster2 = jj;
                        minDis = curDis;
                    }
                }
            }
            // join clusters
            int index = clusterBeg[cluster1];
            //
            while ( indLinks[index] != nr) {
                index = indLinks[index];
            }
            //
            indLinks[index] = clusterBeg[cluster2];
            //
            --nrOfClusters;
            clusterBeg[cluster2] = clusterBeg[nrOfClusters];
        }
        //
        // get centroids
        ParetoIndividual centroids[] = new ParetoIndividual[nrOfClusters];
        //
        for (int ii=0; ii<nrOfClusters; ii++) {
            double minDis = max_double;
            int centroid = 0,
                x = clusterBeg[ii];
            //
            while (x != nr) {
                double curDis = 0.;
                int y = clusterBeg[ii];
                ParetoIndividual ind = tempIndividual[x];
                //
                while (y != nr) {
                    curDis += ind.XDistance(tempIndividual[y]);
                    curDis += ind.TDistance(tempIndividual[y]);
                    y = indLinks[y];
                }
                //
                if (curDis < minDis) {
                    minDis = curDis;
                    centroid = x;
                }
                //
                x = indLinks[x];
            }
            //
            try {
                centroids[ii] = (ParetoIndividual)tempIndividual[centroid].clone();
                gen_num_entered[ii] = temp_gen_num[centroid];
            } catch (CloneNotSupportedException cnse) {

            }
        //
        }
        for (int ii=0; ii<number_of_indiv; ii++) {
            individuals[ii] = null;
            if (centroids[ii] != null) {
                individuals[ii] = centroids[ii];
            }
        }
        extra_individuals = 0;
        return true;
    }
```

```
}
```

D.4.4 Objective Abstract Class

Objective.java

```
package edu.asdl.design.ea.spea;
//
//  Objective.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Wed Sep 24 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//

public abstract class Objective implements Cloneable {
    //
    // Abstract Objective class, used to define the basic methods that the XParetoFitness class
    // will access. Individual Objective classes for each underlying tool will implement this
    // abstract class
    //
    protected double results[], goals[], tracked[];

    public abstract boolean run();

    public abstract double[] getResults();

    public abstract double[] getGoals();

    public abstract double[] getTracked();

    public abstract double[] getFree();

    public abstract double[] getTrackedNormal();

    public abstract double[] getInputs();

    public abstract Object clone() throws CloneNotSupportedException;

    public int getIterationNumber() {
        return 1;
    }

    public abstract void replaceIndividual(ParetoIndividual in);
}
```

D.4.5 X Parameter Pareto Fitness Class

XParetoFitness.java

```
package edu.asdl.design.ea.spea;
//
//  XParetoFitness.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Wed Sep 24 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//

import edu.asdl.design.ea.basic.*;
import edu.asdl.design.ea.basic.Fitness;

public class XParetoFitness extends Fitness implements Cloneable {
    //
    // The Pareot fitness class for MSPEA, calulates the strength in the Parameter, Response, and
    // "Tracked" spaces. This is a major difference between MSPEA and the standard SPEA.
    //
    Objective goal;
    Threaded_Objective thread;
    //
    public XParetoFitness() {}
    //
    // Only Primary constructor shown for brevity
    //
    public XParetoFitness(int number_fit_objects, float[] weights, Objective object) {
        super(number_fit_objects, weights);
        this.goal = object;
    }
    //
    public void placeObjective(Objective object) {} // Code omitted for brevity
    //
    public double[] getTracked() {} // Code omitted for brevity
    //
    public double[] getInputs() {} // Code omitted for brevity
    //
    public double[] getOutputs() {} // Code omitted for brevity
    //
    public double[] getFree() {} // Code omitted for brevity
    //
    public void replaceIndividual(ParetoIndividual in) {} // Code omitted for brevity
    //
```

```
    public Object clone() throws CloneNotSupportedException{} // Clone code omitted for brevity
    //
    public boolean runObjective() {
        //
        // Runs the Objective to update the fitness values
        // Must be run before any domination or strength
        // calculations.
        //
        boolean test = goal.run();
        if (test) {
            double intermediate_results[] = goal.getResults();
            double result_goals[] = goal.getGoals();
            for (int ii=0; ii< objectives; ii++) {
                fitness_indv[ii] = -1.0 * Math.abs((intermediate_results[ii] - result_goals[ii])
                                                   / result_goals[ii]);
            }
        }
        return test;
    }
    //
    public void runThreadedObjective() {
        //
        // Runs the Objective to update the fitness values
        // Must be run before any domination or strength
        // calculations.
        //
        thread = new Threaded_Objective(goal);
        thread.go();
    }
    //
    public boolean getThreadedObjective() {
        boolean test = thread.success();
        goal = thread.returnGoal();
        if (test) {
            double intermediate_results[] = goal.getResults();
            double result_goals[] = goal.getGoals();
            for (int ii=0; ii< objectives; ii++) {
                fitness_indv[ii] = -1.0 * Math.abs((intermediate_results[ii] - result_goals[ii])
                                                   / result_goals[ii]);
            }
        }
        return test;
    }
}
```

D.4.6 Threaded Objective Class

Threaded_Objective.java

```
package edu.asdl.design.ea.spea;
//
//  Threaded_Objective.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Fri Oct 10 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//

public class Threaded_Objective implements Runnable{
    //
    // Simple class to allow multi-threaded operation of the objective function. Decreases runtime
    // by approximately 30% on dual CPU machines.
    //
    Thread thread;
    Objective goal;
    boolean test;

    public Threaded_Objective(Objective goal) {
        this.goal = goal;

    }

    public void go() {
        thread = new Thread(this, "Fitness");
        thread.start();
    }

    public void run() {
         test = goal.run();
    }

    public boolean success() {
        return test;
    }

    public Objective returnGoal() {
        return goal;
    }

}
```

D.4.7 Pareto Tournament Class

ParetoTournement.java

```
package edu.asdl.design.ea.spea;
//
//  ParetoTournement.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Thu Oct 02 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
import java.util.Random;
import edu.asdl.design.ea.basic.*;
import edu.asdl.design.ea.basic.Population;
import edu.asdl.design.ea.basic.SimpleIndividual;
import edu.asdl.design.ea.basic.Tournement;

public class ParetoTournement extends Tournement {
    //
    // The MSPEA specific Tournement selection routine, extends the basic Tournement class.
    //
    ParetoPopulation internalPop, childPop;

    public ParetoTournement(ParetoPopulation pop, int number_of_children) {
        internalPop = (ParetoPopulation)pop;
        internal_length = internalPop.getNumberIndividuals();
        population_number = number_of_children;
        numgenes = internalPop.getNumGenes();
        numbits = internalPop.getNumBits();
        createChildren();
    }

    void createChildren() {
        childPop = new ParetoPopulation(population_number, numgenes, numbits);
    }

    public boolean compete() {
        boolean success = false;
        Random randgen = new Random();
        int popnumber[][] = new int[population_number][2];
        for (int ii=0; ii<population_number; ii++) {
            popnumber[ii][0] = randgen.nextInt(internal_length);
            popnumber[ii][1] = randgen.nextInt(internal_length);
        }
        //
        float str_1, str_2;
        int ind_1, ind_2;
        for (int ii=0; ii < population_number; ii++) {
            ind_1 = popnumber[ii][0];
            ind_2 = popnumber[ii][1];
            str_1 = internalPop.individuals[ind_1].Strength();
            str_2 = internalPop.individuals[ind_2].Strength();
            if (str_1 <= str_2) {
                try {
                    childPop.individuals[ii] =
                    (ParetoIndividual)internalPop.individuals[ind_1].clone();
                } catch (CloneNotSupportedException cnse) {
                }
            } else {
                try {
                    childPop.individuals[ii] =
                    (ParetoIndividual)internalPop.individuals[ind_2].clone();
                } catch (CloneNotSupportedException cnse) {
                }
            }
        }
        return success;
    }

    public Population getChildPop() {
        return childPop;
    }
}
```

D.4.8 Generation Iterator Class

Generations.java

```
package edu.asdl.design.ea.spea;
//
//  Generations.java
//  JAVA SPEA
//
//  Created by Peter Hollingsworth on Thu Oct 02 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
import java.util.Random;
import java.util.*;
import edu.asdl.design.ea.basic.*;
import edu.asdl.design.ea.basic.Population;
//
```

```
public class Generations implements Runnable{
    //
    // The basic runtime of the MSPEA tool, performs the multiple generation loop, and makes the
    // calls to the subordinate classes and methods
    //
    public ParetoPopulation Pop, Pop_History[];
    public ExternalParetoPopulation externalPop;
    int max_generations = 300, external_population = 150, nominal[];
    float crossover_percentage = 0.35f, mutation_percentage = 0.25f;
    public Thread GA_thread;
    int maxX[], minX[], num_param, numtracked;
    double maxXout[], minXout[], nominalTrack[];
    final double maxDouble = Double.MAX_VALUE, minDouble = Double.MIN_VALUE;
    final int maxInt = Integer.MAX_VALUE;
    int maxthreads = 10;
    Random rand_gen;
    ArrayList cross_list;
    Runtime r = Runtime.getRuntime();
    //
    public Generations(ParetoPopulation pop) {
        this.Pop = pop;
        nominal = new int[Pop.getNumberOptParam()];
        maxX = new int[Pop.getNumberOptParam()];
        minX = new int[Pop.getNumberOptParam()];
        numtracked = Pop.getNumberTracked();
        maxXout = new double[numtracked];
        minXout = new double[numtracked];
        nominalTrack = new double[numtracked];
        num_param = Pop.getNumberOptParam();
        external_population = Pop.external_indiv;
        for (int ii=0; ii<num_param; ii++) {
            maxX[ii] = 0;
            minX[ii] = maxInt;
        }
        for(int jj=0; jj<numtracked; jj++) {
            maxXout[jj] = minDouble;
            minXout[jj] = maxDouble;
        }
    }
    //
    public void startThread() {
        GA_thread = new Thread(this, "Loop");
        GA_thread.start();
    }
    //
    // Public Methods for setting and obtaining the EA properties have been omitted for brevity
    //
    public int getExternalPopulationSize() {
        return external_population;
    }
    //
    public ParetoPopulation getInternalPopulation() {
        return Pop;
    }
    //
    public ExternalParetoPopulation getExternalPopulation() {
        return externalPop;
    }
    //
    public ParetoPopulation[] getPopulationHistory() {
        return Pop_History;
    }
    //
    public void putMaxThreads(int thds) {
        maxthreads = thds;
    }
    //
    public void run() {
        // The method to ensure that the EA runs in a seperate thread
        externalPop = new ExternalParetoPopulation(external_population, Pop);
        rand_gen = new Random();
        loop();
        System.out.println("Generations are Done.");
    }

    void loop() {
        boolean added_to_external[] = new boolean[Pop.number_of_indiv];
        int genevalues[][] = new int[Pop.number_of_indiv][nominal.length];
        double trackedvalues[] = new double[numtracked];
        //
        // Evaluate nominal input X point
        //
        int numproc = r.availableProcessors();
        if (numproc > maxthreads) {
            numproc = maxthreads;
        }
        System.out.println("Using " +numproc +" objective function threads.");
        //
        int remain = Pop.number_of_indiv % numproc;
        int loopbase = Pop.number_of_indiv - remain;
        for (int ii=0; ii<loopbase; ii+= numproc) {
```

```
        for (int jj=ii; jj<ii+numproc; jj++) {
            Pop.individuals[jj].realChrome.update(Pop.individuals[jj].bitChrome);
            genevalues[jj] = Pop.individuals[jj].realChrome.getGeneValues();
            Pop.individuals[jj].indivFit.runThreadedObjective();
        }
        for (int jj=ii; jj<ii+numproc; jj++) {
            try{
                Pop.individuals[jj].indivFit.thread.thread.join();
            } catch (InterruptedException ie) {
            }
            Pop.individuals[jj].fitnessEval =
            Pop.individuals[jj].indivFit.getThreadedObjective();
        }
    }
    // Mop-up loop to catch remainder of the populaiton is omitted for brevity
    //
    for (int ii=0; ii<Pop.number_of_indiv; ii++) {
        for (int jj=0; jj<nominal.length; jj++) {
            if (genevalues[ii][jj] > maxX[jj]) {
                maxX[jj] = genevalues[ii][jj];
            }
            if (genevalues[ii][jj] < minX[jj]) {
                minX[jj] = genevalues[ii][jj];
            }
        }
        trackedvalues = Pop.individuals[ii].indivFit.getTracked();
        for (int kk=0; kk<numtracked; kk++) {
            if (trackedvalues[kk] > maxXout[kk]) {
                maxXout[kk] = trackedvalues[kk];
            }
            if (trackedvalues[kk] < minXout[kk]) {
                minXout[kk] = trackedvalues[kk];
            }
        }
    }
    for (int jj = 0; jj<nominal.length; jj++) {
        nominal[jj] = (maxX[jj] + minX[jj]) >> 1;
    }
    for (int kk=0; kk<numtracked; kk++) {
        nominalTrack[kk] = 0.5*(maxXout[kk] + minXout[kk]);
    }
    //
    // Determine domination
    //
    for (int ii=0; ii<Pop.number_of_indiv-1; ii++) {
        for (int jj = ii+1; jj<Pop.number_of_indiv; jj++) {
            if (Pop.dominates(ii,jj, nominal, nominalTrack)) {
                Pop.individuals[jj].isDominated = true;
            } else if (Pop.dominates(jj, ii, nominal, nominalTrack)) {
                Pop.individuals[ii].isDominated = true;
                break;
            }
        }
    }
    //
    // Fill External Pareto Population
    //
    boolean status[] = Pop.getDominationStatus();
    boolean xstatus[] = Pop.getXDominationStatus();
    boolean ystatus[] = Pop.getYDominationStatus();
    boolean tstatus[] = Pop.getTDominationStatus();
    boolean good;
    for (int ii=0; ii<Pop.number_of_indiv; ii++) {
        Pop.individuals[ii].realChrome.update(Pop.individuals[ii].bitChrome);
        Pop.individuals[ii].indivFit.replaceIndividual(Pop.individuals[ii]);
        good = Pop.individuals[ii].good;
        if ((!status[ii]) & (good)) {
            added_to_external[ii] =
            externalPop.copyTo(Pop.individuals[ii], nominal, nominalTrack,0);
        }
    }
    if (externalPop.extra_individuals != 0) {
        added_to_external[0] = externalPop.cluster();
    }
    //
    // Setup Complete now start the generations
    //
    int generation = 0, ii, total_indiv,
        gene_swap,  cross_1, cross_2;
    short gene_mut, bit_mut;
    ParetoPopulation TourneyPop, ChildPop, CrossPop, MutPop;
    ChildPop = new ParetoPopulation(Pop.number_of_indiv, Pop.numgenes, Pop.numbits);
    CrossPop = new ParetoPopulation(Pop.number_of_indiv, Pop.numgenes, Pop.numbits);
    MutPop = new ParetoPopulation(Pop.number_of_indiv, Pop.numgenes, Pop.numbits);
    ParetoTournement Tourney;
    boolean competed, crossed, mutated;
    //
    // Tournement Selection, produces survivor child population
    //
    float indivStrength[];
```

```
        while (generation < max_generations) {
            TourneyPop = new ParetoPopulation(Pop.number_of_indiv +
                                              externalPop.countMembers(),
                                              Pop.numgenes, Pop.numbits);
            indivStrength = new float[Pop.number_of_indiv + externalPop.countMembers()];
            for (ii=0; ii<Pop.number_of_indiv; ii++) {
                TourneyPop.individuals[ii] = Pop.individuals[ii];
            }
            for (int jj=0; jj<externalPop.countMembers(); jj++) {
                TourneyPop.individuals[jj+ii] = externalPop.individuals[jj];
            }
            total_indiv = Pop.number_of_indiv + externalPop.countMembers();
            for (ii=0; ii< total_indiv; ii++) {
                TourneyPop.XStrength(TourneyPop.individuals[ii], nominal);
                TourneyPop.YStrength(TourneyPop.individuals[ii]);
                TourneyPop.TStrength(TourneyPop.individuals[ii], nominalTrack);
                indivStrength[ii] = TourneyPop.individuals[ii].Strength();
            }
            //
            Tourney = new ParetoTournement(TourneyPop, Pop.number_of_indiv);
            competed = Tourney.compete();
            ChildPop = (ParetoPopulation)Tourney.getChildPop();
            //
            // Crossover
            //
            cross_list = new ArrayList(ChildPop.number_of_indiv);
            for (int kk=0; kk<ChildPop.number_of_indiv; kk++) {
                try {
                    CrossPop.individuals[kk] = (ParetoIndividual)ChildPop.individuals[kk].clone();
                } catch (CloneNotSupportedException cnse) {
                }
                cross_list.add(new Integer(kk));
            }
            //
            Collections.shuffle(cross_list, rand_gen);
            Object crossArray[] = cross_list.toArray();
            for (int kk=0; kk<ChildPop.number_of_indiv-1; kk+=2) {
                cross_2 = ((Integer)crossArray[kk+1]).intValue();
                cross_1 = ((Integer)crossArray[kk]).intValue();
                if (rand_gen.nextFloat() <= crossover_percentage) {
                    //
                    // Select Crossover type
                    //
                    gene_swap = rand_gen.nextInt(CrossPop.individuals[kk].bitChrome.gene.length);
                    if (rand_gen.nextInt(2) == 1) {
                        CrossPop.individuals[cross_2].bitChrome.gene[gene_swap] =
                        CrossPop.individuals[cross_1].bitChrome.tradeGene((short)gene_swap,
                                CrossPop.individuals[cross_2].bitChrome.gene[gene_swap]);
                    } else {
                        CrossPop.individuals[cross_2].bitChrome.gene[gene_swap] =
                        CrossPop.individuals[cross_1].bitChrome.recombineGenes((short)gene_swap,
                                    CrossPop.individuals[cross_2].bitChrome.gene[gene_swap]);
                    }
                    CrossPop.individuals[cross_1].fitnessEval = false;
                    CrossPop.individuals[cross_2].fitnessEval = false;
                }
            }
            //
            // Mutation
            //
            for (int kk=0; kk<CrossPop.number_of_indiv; kk++) {
                try {
                    MutPop.individuals[kk] = (ParetoIndividual) CrossPop.individuals[kk].clone();
                } catch (CloneNotSupportedException cnse) {
                }
                if (rand_gen.nextFloat() <= mutation_percentage) {
                    gene_mut = (short)(rand_gen.nextInt(
                                            MutPop.individuals[kk].bitChrome.getGeneNumber()));
                    bit_mut = (short)
                        (rand_gen.nextInt(MutPop.individuals[kk].
                                    bitChrome.getBitNumber((short)gene_mut)));
                    MutPop.individuals[kk].bitChrome.flipBit(gene_mut, bit_mut);
                    MutPop.individuals[kk].fitnessEval = false;
                }
                MutPop.individuals[kk].isDominated = false;
            }
            //
            // Place previous generation in storage,
            // replace Pop with MutPop
            //
            for (int kk=0; kk<Pop.number_of_indiv; kk++) {
                try {
                    Pop.individuals[kk]= (ParetoIndividual)MutPop.individuals[kk].clone();
                } catch (CloneNotSupportedException cnse) {
                }
            }
            generation++;
            //
            // Reevaluate the nominal X point
```

```
            //
            for (int jj=0; jj<num_param; jj++) {
                maxX[jj] = 0;
                minX[jj] = maxInt;
            }
            for(int jj=0; jj<numtracked; jj++) {
                maxXout[jj] = minDouble;
                minXout[jj] = maxDouble;
            }
            //
            // Recalculate fitness if Necssary
            //
            for (ii=0; ii<loopbase; ii+= numproc) {
                for (int zz=ii; zz<ii+numproc; zz++) {
                    Pop.individuals[zz].realChrome.update(Pop.individuals[zz].bitChrome);
                    Pop.individuals[zz].indivFit.replaceIndividual(Pop.individuals[zz]);
                    genevalues[zz] = Pop.individuals[zz].realChrome.getGeneValues();
                    if (!Pop.individuals[zz].fitnessEval) {
                        Pop.individuals[zz].indivFit.runThreadedObjective();
                    }
                }
                for (int jj=ii; jj<ii+numproc; jj++) {
                    if (!Pop.individuals[ii].fitnessEval) {
                        try{
                            Pop.individuals[jj].indivFit.thread.thread.join();
                        } catch (InterruptedException ie) {
                        }
                        Pop.individuals[jj].fitnessEval =
                        Pop.individuals[jj].indivFit.getThreadedObjective();
                    }
                }
            }
            //
            // Mop Up Loop, to catch the remainder of the population is omitted for brevity
            //
            for (ii=0; ii<Pop.number_of_indiv; ii++) {
                for (int jj=0; jj<nominal.length; jj++) {
                    if (genevalues[ii][jj] > maxX[jj]) {
                        maxX[jj] = genevalues[ii][jj];
                    }
                    if (genevalues[ii][jj] < minX[jj]) {
                        minX[jj] = genevalues[ii][jj];
                    }
                }
            }
            for (int jj = 0; jj<nominal.length; jj++) {
                nominal[jj] = (maxX[jj] + minX[jj]) >> 1;
            }
            for (int kk=0; kk<numtracked; kk++) {
                nominalTrack[kk] = 0.5*(maxXout[kk] + minXout[kk]);
            }
            //
            // Redetermine Domination
            //
            for (int kk=0; kk<Pop.number_of_indiv-1; kk++) {
                for (int jj = ii+1; jj<Pop.number_of_indiv; jj++) {
                    if (Pop.dominates(kk,jj, nominal, nominalTrack)) {
                        Pop.individuals[jj].isDominated = true;
                    } else if (Pop.dominates(jj, kk, nominal, nominalTrack)) {
                        Pop.individuals[kk].isDominated = true;
                        break;
                    }
                }
            }
            //
            // Add Points to the External Pareto pop
            //
            status = Pop.getDominationStatus();
            for (int kk=0; kk<Pop.number_of_indiv-1; kk++) {
                Pop.individuals[kk].realChrome.update(Pop.individuals[kk].bitChrome);
                Pop.individuals[kk].indivFit.replaceIndividual(Pop.individuals[kk]);
                good = Pop.individuals[kk].good;
                if ((!status[kk]) & (good)) {
                    added_to_external[kk] = externalPop.copyTo(Pop.individuals[kk], nominal,
                                                               nominalTrack, generation);
                }
            }
            if (externalPop.extra_individuals != 0) {
                added_to_external[0] = externalPop.cluster();
            }
            System.out.println("Generation: " +generation +" done.");
        } // Bottome of the grans "while loop"
    }
}
```

APPENDIX E

EXAMPLE MSPEA INPUT FILE

Input.in

```
Type
type SMR

Inputs
FD.number_crew short 1 2 1
FD.number_pax short 0 15 4
CWD.crew_weight int 200 300 4
CWD.passenger_weight int 150 450 4
FD.amarment_weight float 0 2500 4
MRD.pressure_altitude short 0 6000 4
MRD.delta_temp float 0 60 4
MRD.NML float 2 5 4
TRD.NML float 2 5 4
PD.number_engines int 1 2 1
PD.contingency_rating float 25 75 4
MD.dash_speed int 150 300 4
WD.dive_speed int 150 300 4
MRD.download_perc float 2.3 4.2 4
PD.fixed_losses float 10 100 4
MD.speed[2] int 120 160 4
MD.speed[3] int 0 100 4
MD.speed[5] int 0 100 4
MD.speed[7] int 100 140 4
MD.speed[11] int 120 160 4
MD.speed[12] int 0 100 4
MD.speed[16] int 0 100 4
MD.speed[17] int 120 160 4
MD.speed[19] int 120 160 4
MD.time[2] float 10 30 4
MD.time[3] float 0 30 4
MD.time[4] float 0 20 4
MD.time[5] float 0 15 4
MD.time[6] float 0 30 4
MD.time[7] float 0 10 4
MD.time[8] float 0 1 1
MD.time[9] float 0 7 3
MD.time[10] float 0 1 1
MD.time[11] float 0 30 4
MD.time[12] float 0 30 4
MD.time[13] float 0 1 1
MD.time[14] float 0 7 3
MD.time[15] float 0 1 1
MD.time[16] float 5 30 4
MD.time[17] float 10 30 4
MD.time[19] float 20 40 4

Outputs
ED.sfc[0] 0.490
ED.sfc[1] 0.490
ED.sfc[2] 0.490
ED.sfc[3] 0.501
ED.sfc[4] 0.531
ED.sfc[5] 0.763
ED.power_fraction[2] 88
ED.power_fraction[3] 73
ED.power_fraction[4] 55
ED.power_fraction[5] 10
ED.time[1] 2.5
ED.hp[0] 1196
ED.hp[1] 957
ED.hp[2] 840
ED.hp[3] 702
```

```
ED.hp[4] 527
ED.hp[5] 96
PD.SFC_hover 0.490
PD.XMSN_eff 89
VSD.MR_VTIP 700
TD.bare_engine_K 0.89
TD.MR_blades_K 0.9
TD.MR_hub_K 0.8
TD.fuselage_K 0.8
TD.wing_K 0.8
TD.TR_blades_K 0.85
TD.TR_hub_K 0.85
TD.h_tail_K 0.7
TD.v_tail_K 0.7
TD.powerPlant_section_K 0.8
TD.transmission_K 0.89

Tracked
GW_out        0
HP_out        0

Free
WD.wing_number int 0 1 1
ED.sfc[0] float 0.480 0.510 4
ED.sfc[1] float 0.480 0.510 4
ED.sfc[2] float 0.480 0.510 4
ED.sfc[3] float 0.490 0.530 4
ED.sfc[4] float 0.510 0.550 4
ED.sfc[5] float 0.720 0.800 4
ED.power_fraction[2] float 80 95 3
ED.power_fraction[3] float 65 80 3
ED.power_fraction[4] float 45 60 3
ED.power_fraction[5] float 0 20 3
ED.time[1] float 1 5 4
ED.hp[0] int 1000 2000 4
ED.hp[1] int 800 1800 4
ED.hp[2] int 600 1500 4
ED.hp[3] int 500 1400 4
ED.hp[4] int 300 1000 4
ED.hp[5] int 0 400 4
PD.SFC_hover float 0.480 0.510 4
PD.XMSN_eff float 80 97 4
TD.MR_blades_K float 0.5 1.1 5
TD.MR_hub_K float 0.5 1.1 5
TD.wing_K float 0.5 1.1 5
TD.TR_blades_K float 0.5 1.1 5
TD.TR_hub_K float 0.5 1.1 5
TD.h_tail_K float 0.5 1.1 5
TD.v_tail_K float 0.5 1.1 5
TD.fuselage_K float 0.5 1.1 5
TD.powerPlant_section_K float 0.5 1.1 5
TD.transmission_K float 0.5 1.1 5
MRD.tip_Mach float .5 .9 4
MRD.solidity float 0.02 0.12 5
MRD.blade_number short 2 9 3
MRD.disc_loading float 5 20 5
MRD.root_cut_out float 0 10 4
MRD.number short 1 2 1
MRD.tcMR float 5 15 4
TRD.solidity float 0.05 0.3 5
TRD.blade_number short 2 9 3
TRD.disc_loading float 5 20 5
TRD.root_cut_out float 0 10 4
TRD.tcTR float 5 15 3
WD.wing_span int 0 40 4
WD.wing_area int 0 250 4
WD.GW_on_wing float 0 80 7
WD.taper_ratio float 0.5 1 4
WD.tcWING float 8 20 4
WD.wing_incidence_angle float -5 10 4
HTD.b_HT float 5.0 25.0 4
HTD.P_G float 0.05 0.30 4
HTD.R_HT float 0.01 2 4
HTD.taper_HT float 0.75 1 3
HTD.S_HT float 10 150 5
HTD.tc_HT float 5 20 4
VTD.b_VT float 5 15 4
```

```
VTD.S_VT float 25 60 4
VTD.Y_G float 0.05 0.30 4
VTD.h_VT float 0.01 4 4
VTD.taper_VT float 0.5 1 3
VTD.tc_VT float 5 20 4
FD.de_icing_equipment float 10 200 5
PD.SKPDSZ int 2 3 1
PD.fuel_tankage_ratio float 0.05 0.1 3
PD.SKT float 1 1.3 1
```

APPENDIX F

FUEL RATIO SIZING METHOD

The Fuel Ratio ($\mathbf{R_f}$) sizing method, which provided the basic tool for the MSPEA validation and example cases is a relatively simple tool, that in the general sense operates at a high level of abstraction. However, this is not inherently a problem as any tool based upon the $\mathbf{R_f}$ method can be expanded to increase the level of fidelity as necessary. Furthermore, using the hierarchical decomposition described in Hypothesis 2 allows injection of medium levels of fidelity without necessarily increasing the overall complexity of the tool. This ultimately can be used to allow the engineer/designer to include the appropriate amount of knowledge early on in the design process.

F.1 Description of the $\mathrm{R_f}$ *Method*

The $\mathbf{R_f}$ method is an energy based sizing method designed to achieve a fuel balance across an aircraft/rotorcraft mission. Primarily used, in a graphical manner, by the helicopter industry in the 1950s-1970s, $\mathbf{R_f}$ is the ratio of fuel weight to gross weight of the vehicle [128, 129]. The method is still used today in the conceptual design of rotorcraft, and other VTOL aircraft. Furthermore, it is inherently related to the aircraft design methods taught by Roskam, Raymer, or Mattingly in their design textbooks [130, 131, 132].

The basic premise of the method is that two $\mathbf{R_f}$ equations, the aerodynamic (required) $\mathbf{R_f}$ and weight (available) $\mathbf{R_f}$ are matched for a certain set of design parameters, $\mathbf{X} = (x_1, x_2, x_3, \ldots, x_n)$, range desired and performance criteria [129]. The $\mathbf{R_f}$ required equation is given in Equation 53 [129].

$$\mathrm{R_f} = \triangle \mathrm{R_{f_{sw}}} + \triangle \mathrm{R_{f_{cl}}} + \triangle \mathrm{R_{f_{cr}}} + k_r \triangle \mathrm{R_{f_r}} \tag{53a}$$

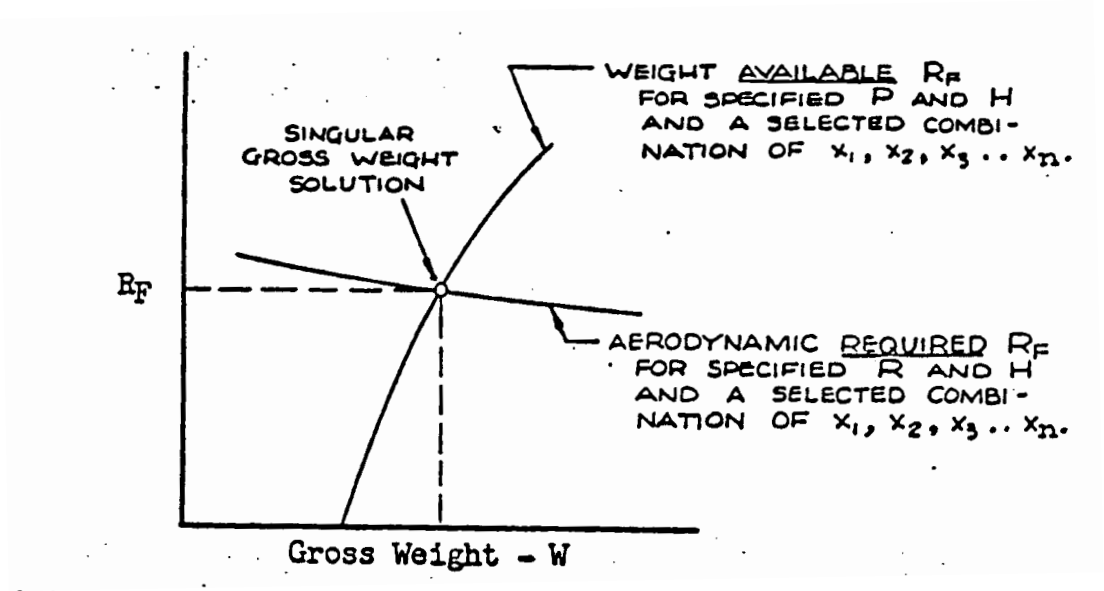


Figure 97: Single Gross Weight $\mathbf{R_f}$ Solution [129]

$$\mathrm{R_f} = f\left(\mathrm{W}, \mathrm{H}, \mathrm{R}, x_1, x_2, x_3, \ldots, x_n\right) \tag{53b}$$

Where H is the performance criterion, W is the gross weight, and R the range. The available $\mathbf{R_f}$ is given in Equation 54 [128].

$$\mathrm{R_f} = \frac{W_f}{W} = \frac{[(W_{e1} + W_{e2} + W_{e3} + \ldots + W_{en}) + W_c + P + W_F]}{W_f} \tag{54a}$$

$$\mathrm{R_f} = f\left(\mathrm{W}, \mathrm{P}, x_1, x_2, x_3, \ldots, x_m\right) \tag{54b}$$

Where the W_{ei} are component weights, P is the payload, W_c the crew weight, and W_F the fuel weight. If both forms of the $\mathbf{R_f}$ are equated and a single locus of a closed gross weight solution is found. This is represented in Figure 97.

By solving for several loci, and plotting them on the gross weight, W, vs. $\mathbf{R_f}$ axis, it is then possible to determine the "solution locus curve." The left most point of which represents the minimum gross weight [129]. This is shown in Figure 98. Incidentally, the graphical method was used by Hiller and others because of the lack of computational power available at the time. By solving for only a few points, the graphic allows for easy pattern recognition, and a satisfactory solution to the design

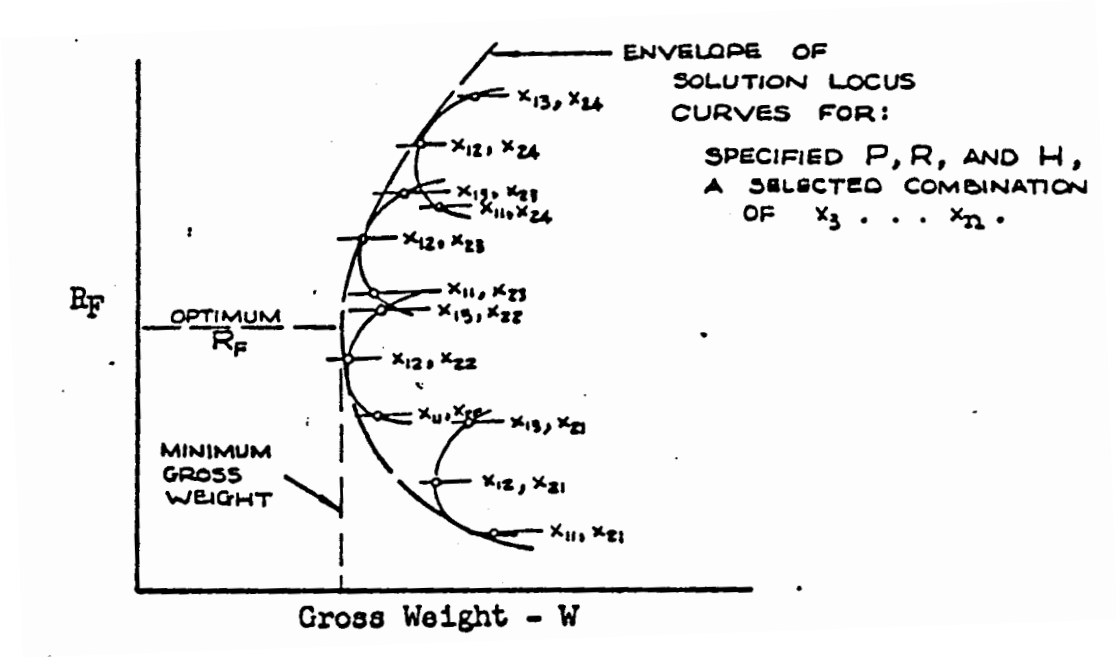


Figure 98: Solution Locus Curves for the $\mathbf{R_f}$ Method [129]

problem at hand. However, since the 1970s an explosion of computing power has occurred, making it feasible to perform the solutions to the $\mathbf{R_f}$ method in a purely computational manner.

F.2 Georgia Tech Computational Implementation of the $\mathrm{R_f}$ *Method*

The need to bring the $\mathbf{R_f}$ method to a point where it can be used for rapid design studies, similar to those performed using aircraft sizing codes such as FLOPS has led Georgia Tech to implement the $\mathbf{R_f}$ method in a computer spreadsheet. A screenshot of the Georgia Tech Rotorcraft Sizing Program spreadsheet is shown in Figure 99. This program can be used to size, single main rotor, tandem rotor, and coaxial helicopters and tiltrotors.

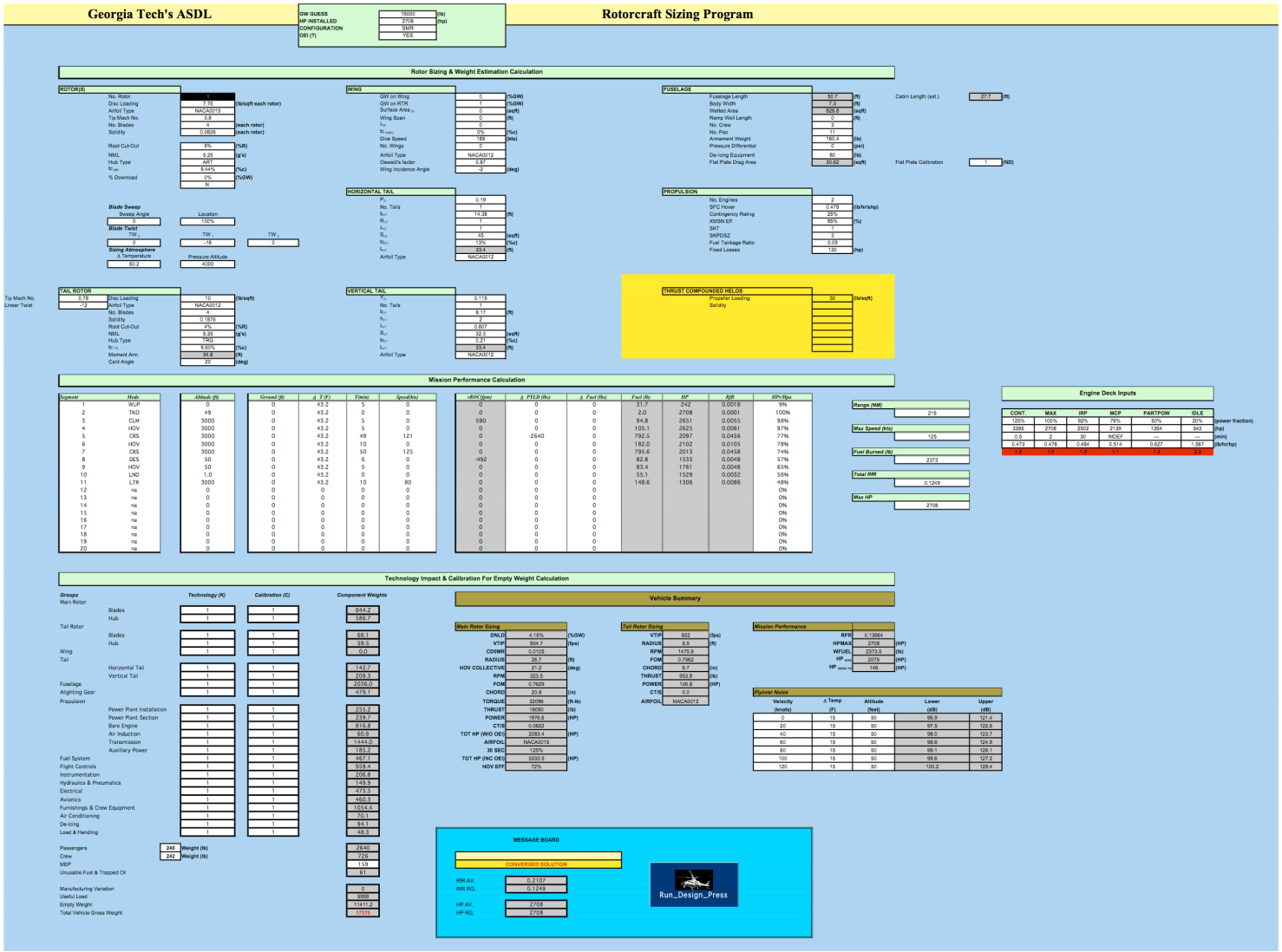


Figure 99: Screenshot of the Georgia Tech $\mathbf{R_f}$ Spreadsheet

F.2.1 Modifications for Incorporation into Technology Boundary Discovery Methods

While the Excel implementation of the $\mathbf{R_f}$ method allows a relatively easy interaction between the engineer and the tool itself, there were several compromises that inhibit its direct use in the RCD design process. The original implementation of the $\mathbf{R_f}$ spreadsheet, was not setup to allow easy batch operations. This stemmed from several problems, including:

- The internal iterator was highly sensitive to the starting guesses for gross-weight and installed horsepower. Convergence time and stability were highly dependent upon these initial guesses
- The internal iterator was particularly slow, often taking several hundred iterations to reach a closed solution
- The computational time for each iteration was significant. This stemmed from the fact that a large percentage of the mathematical calculations were implemented in Visual Basic for Applications (VBA) instead of the Excel front end.

Each of these items necessitated changes to the $\mathbf{R_f}$ tool in order for it to be incorporated into the MSPEA validation work.

F.2.1.1 Changes to Improve Runtime Operation

The foremost change was to modify the iteration scheme to decrease its runtime and increase its stability. The original iterator was replaced with a relaxed fixed point iteration scheme, that operates on the gross-weight & horsepower guesses. Convergence is checked by comparing both the $\mathbf{R_f}$ available vs. $\mathbf{R_f}$ required and the horsepower available vs. horsepower required. Interestingly, the switch to the fixed point iterator solved both the iterator's runtime and stability problems. Furthermore, because the

stability was so significantly enhanced; it was determined that no relaxation was necessary. However, the relaxation factor was included in case stability problems arise in the future.

Unfortunately, the replacement of the iterator does not address the problem with the actual computational speed of each iteration. The interpreted nature of the VBA code. Further, internal compilation of the VBA in Excel did not significantly decrease the computation time. It was, therefore, decided to recode the computations. Because of the nature of the calculation algorithms it was determined that redesigning the algorithm to match the spreadsheet method natively inherent to Excel would be too time consuming. Therefore, a port to a compiled programing language was undertaken.

The choice of programming languages for the port of the $\mathbf{R_f}$ method spreadsheet to what became the $\mathbf{R_f}$ tool was relatively straightforward. Because the MSPEA tool was coded in Java, it was decided that the $\mathbf{R_f}$ tool would also be coded in Java. Further, to simplify the inclusion of the $\mathbf{R_f}$ tool into the MSPEA runtime environment and minimize the runtime of each function evaluation, which is beneficial when several thousand function calls are involved, the $\mathbf{R_f}$ tool was coded in a library form. This eliminated the need for individual input and output (IO) files to be created, read, and parsed. Since IO operations were the time limiting factor in the RAMSCRAM evaluations used in the simple GA described in Appendix C, any time savings accrued through the avoidance of IO files was bound to be significant. Furthermore, because of the library nature of the new $\mathbf{R_f}$ tool, it is possible to easily include the tools functionality in other command line and graphical applications.

Overall, the recoding of the $\mathbf{R_f}$ spreadsheet from Excel to the $\mathbf{R_f}$ tool in Java and the redesign of the internal iterator produced a single function call runtime savings of over two orders of magnitude, from tens of seconds to fractions of a second. This time savings translates into a decrease in runtime for the MSPEA tool running on

top of the $\mathbf{R_f}$ tool from several days to several hours.

F.2.1.2 Changes to Increase Functionality

One of the limiting factors in helicopter design is the effect of high forward speed on the dynamics of the rotor. As the vehicle travels faster the power required diverges from the standard cubic increase. This stems from two conditions. Mach induced drag divergence on the advancing blade, and stall on the retreating blade. The $\mathbf{R_f}$ spreadsheet did not include corrections for either of these phenomena in its original form. Since dash speed was one of the requirements for the LHX program it was necessary to include this capability in the $\mathbf{R_f}$ tool.

The addition of dash speed power required calculations was achieved by including the routines for power required due to compressibility and stall from the GTPDP helicopter sizing program [133]. GTPDP is a more sophisticated sizing tool, with a corresponding increase in the amount of initial knowledge required to use. This increase in complexity only increases the combinatorial complexity of the boundary discovery process, without adding significantly to the understanding derived. Therefore, it was decided to only include a small fraction of the GTPDP functionality instead of replacing the $\mathbf{R_f}$ tool entirely.

APPENDIX G

ADDITIONAL VALIDATION RESULTS

Additional results for the different vehicle types that were investigated for the LHX validation case are contained within this appendix. This is done to minimize the length of Chapter 6.

G.1 Visualization Meta-Modeling Training Verification

The results for verification of the visualization meta-models for the SMR, TDM, and TLTR vehicle types are given below.

G.1.1 Single Main Rotor Helicopter Feasibility Radial Cross Section

A similar graph can be made for the SMR helicopter's response. This plot is shown in Figure 100. This figure can be read in the same manner as the results shown in Figure 46. One area of potential concern, is the fact that the predicted pseudo-function value drops very close to one inside the know feasible region. This could lead to incorrect predictions for values slightly off of the vector presented here. By examining other vectors this was found not to be the case, and the current meta-model was considered acceptable for use in prediction of the feasible regions.

G.1.2 Tandem Rotor Helicopter Feasibility Radial Cross Section

The corresponding chart for the TDM helicopter is shown in Figure 101. The results show that the meta-model produces pseudo-function values significantly greater than one for the feasible region and less than one for the infeasible region. This is ideal for use in the visualization scheme.

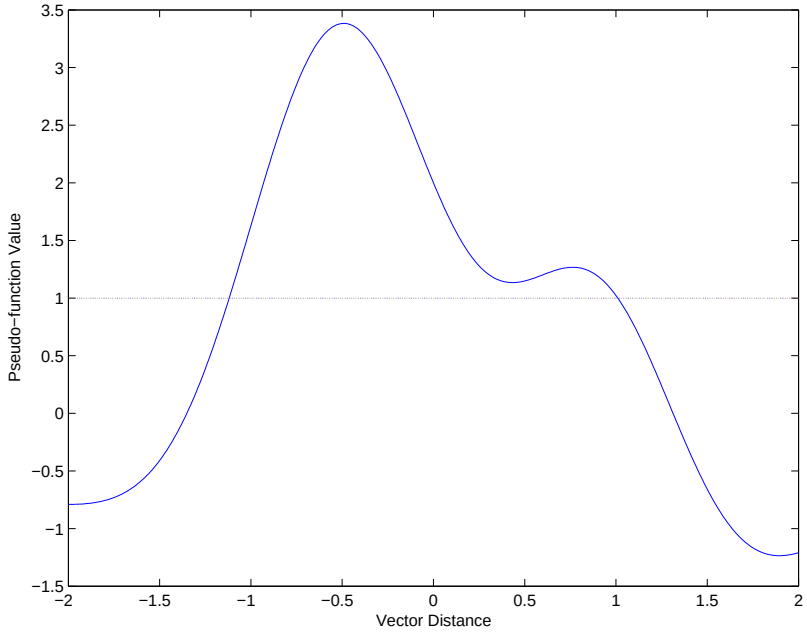


Figure 100: Single Main Rotor Model Radial Cross Section

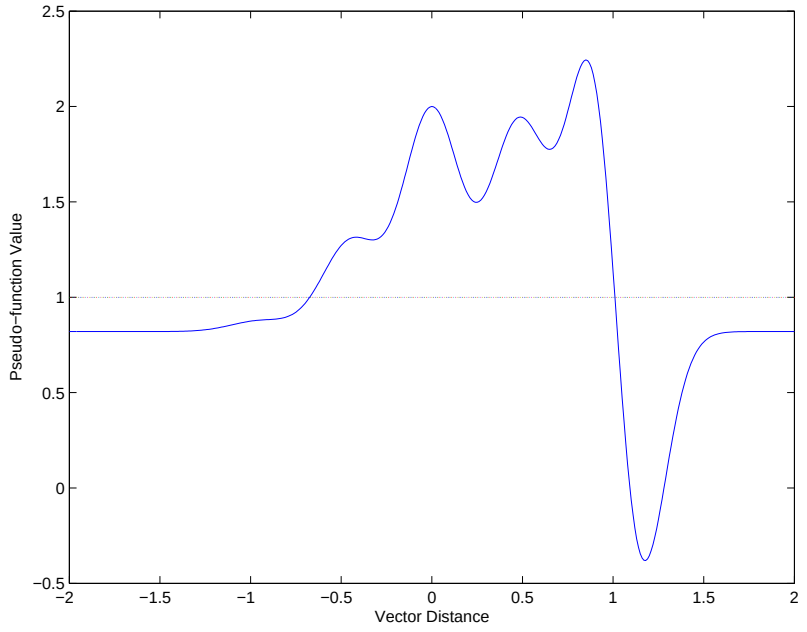


Figure 101: Tandem Rotor Model Radial Cross Section

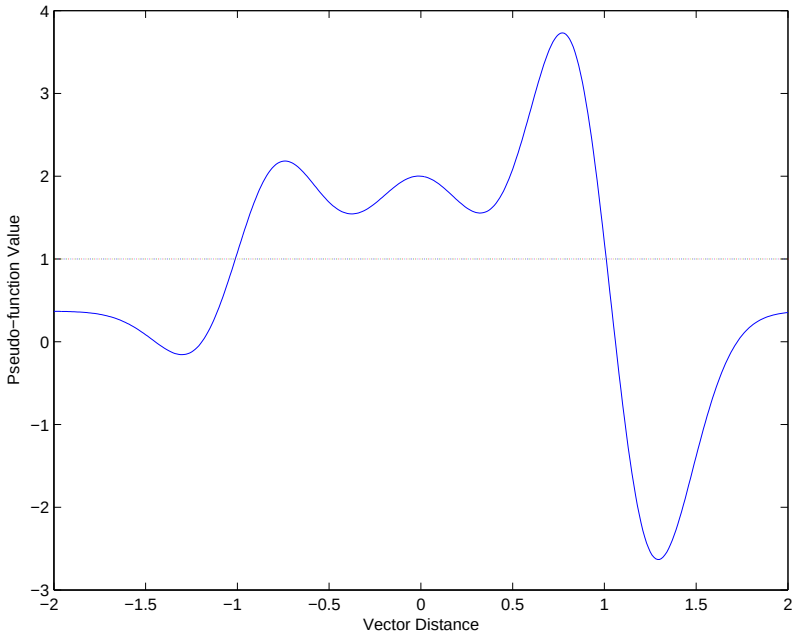


Figure 102: Tiltrotor Model Radial Cross Section

G.1.3 Tiltrotor Feasibility Radial Cross Section

The TLTR meta-model for the 1983 LHX combined, shown in Figure 102, is substantially similar to those for the other vehicle types. This is to be expected.

G.2 Original 1983 LHX Results

The results for the investigation of the COAX, TDM, and TLTR vehicle types for the original 1983 LHX requirements and technology boundaries are given below. Comparing the results to those shown in Section 6.3.3 to those presented below allows the reader to understand the different behaviors of the different vehicle types in greater detail.

G.2.1 Coaxial Rotor Vehicle Type

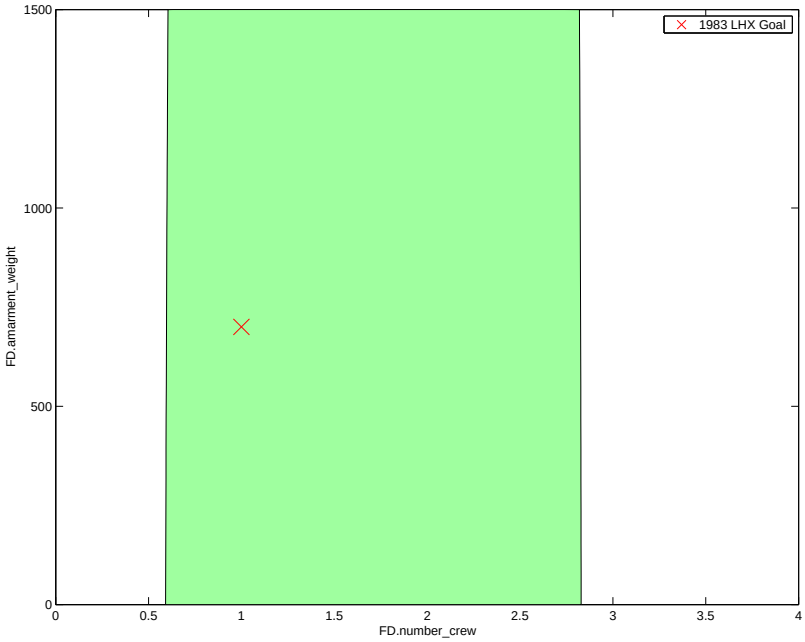
The results for the COAX vehicle type, are shown for the LHX requirements in Figure 103, indicate that a feasible solution region exists for the 1983 LHX reconnaissance mission requirements. The behavior of the COAX system can be compared with that

of the SMR vehicle type by comparing Figures 49 and 50 with Figure 103. Also of interest is the sensitivity of the feasible space to several of the key LHX design requirements have on the COAX vehicle. These are the same ones for which the sensitivities of the SMR vehicle were display in Figure 51. The COAX sensitivity chart is shown in Figure 104.

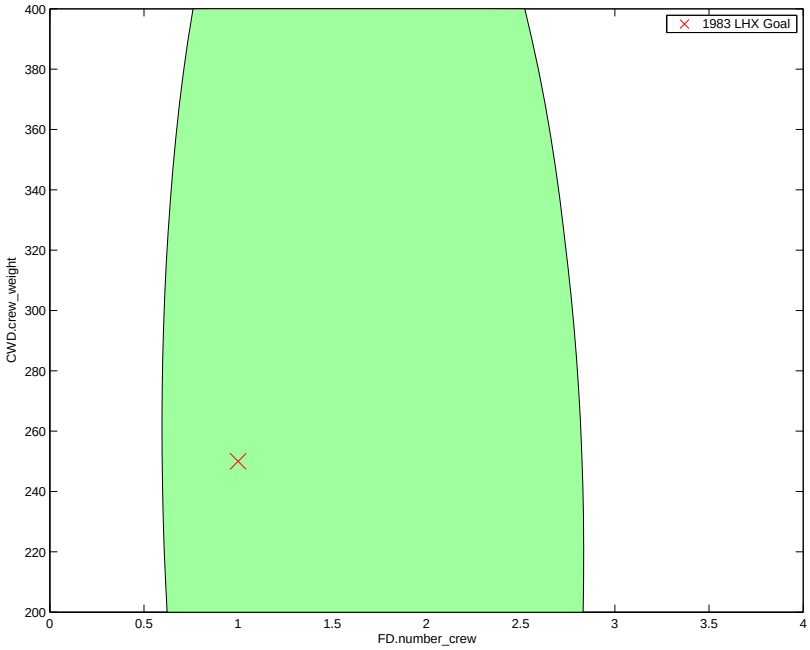
G.2.2 Tandem Rotor Vehicle Type

The TDM is another vehicle type that was not typically studied for the LHX program. Typically TDM designs are used for larger, transport type helicopters. Since the LHX was a light helicopter program, with a major emphasis on combat operations, TDM designs were typically discarded early on in the program. However, since the $\mathbf{R_f}$ tool handles TDM systems, it was relatively trivial to explore the requirements (hyper)space for a theoretical TDM version of the LHX concept. Again an investigation of the requirements space about two planes taken at the original 1983 LHX requirements, shown in Figure 105, shows a feasible space. Of note is the "V" shaped spike in Figure 105(b), this indicates that the MSPEA may have missed a small region of the feasible space. If a requirements setting in this region is desired it is possible to investigate a multitude of states in this region to determine if feasible solutions exist. The specifics of this type of investigation are discussed later in this chapter.

Since the TDM helicopter is able to fulfill the 1983 requirements settings for the LHX system, Figure 106 shows the same sensitivity of the feasible region in the requirements (hyper)space that was displayed in Figure 51. Again from these charts it is possible to get an idea of the types of improvements that are needed to meet greater than LHX requirements.



(a) Number of Crew vs. Armament Weight



(b) Number of Crew vs. Crew Member Weight

Figure 103: COAX LHX Slice in the Requirements Hyperspace, 1983 Settings

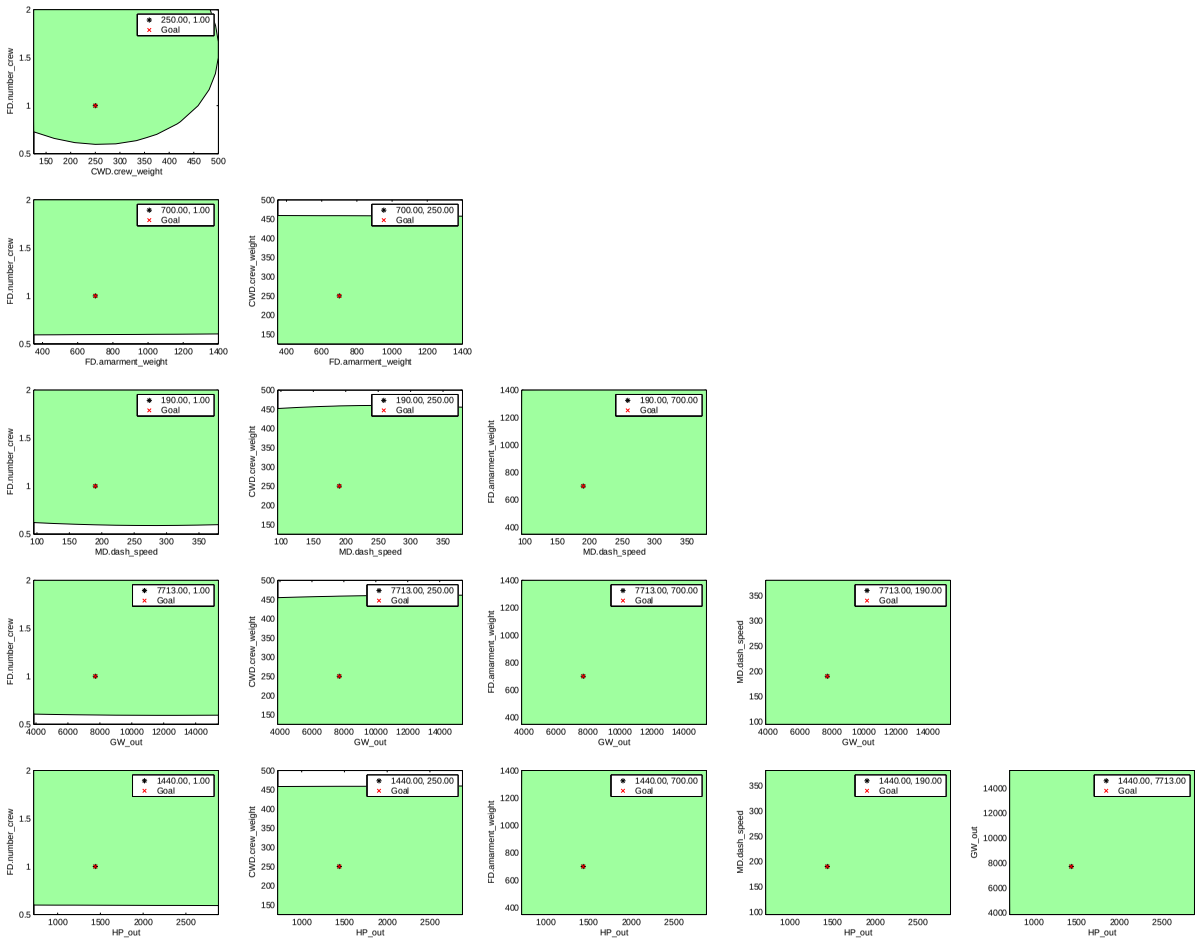
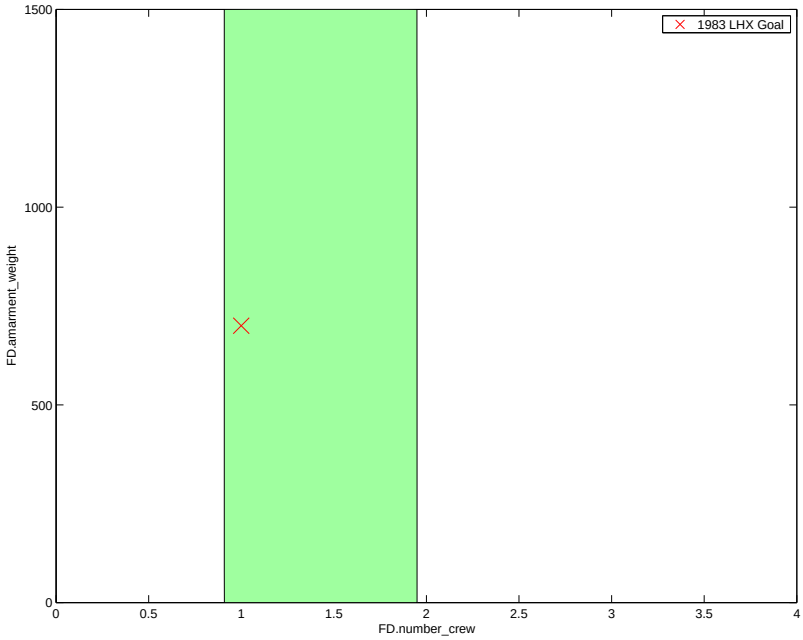
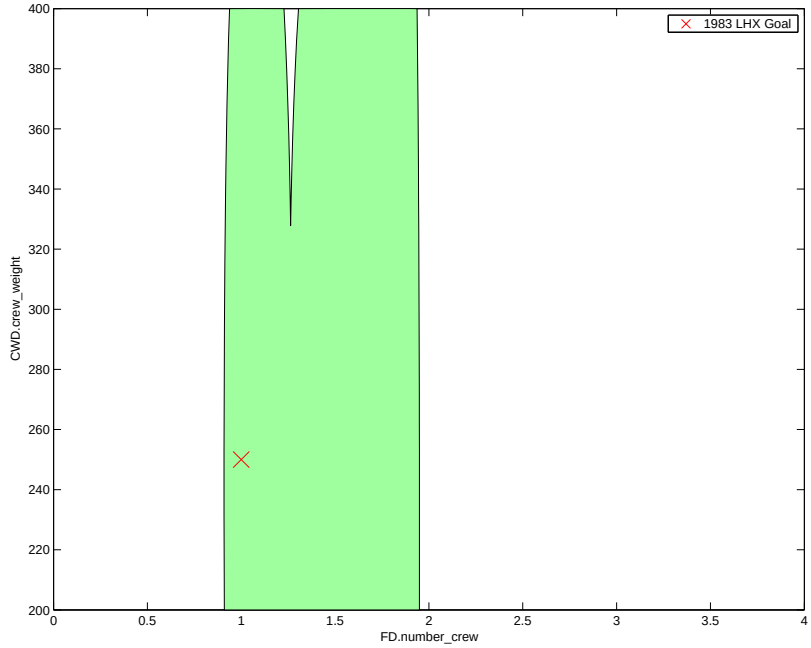


Figure 104: COAX Requirements Feasibility Region, 1983 Settings



(a) Number of Crew vs. Armament Weight



(b) Number of Crew vs. Crew Member Weight

Figure 105: TDM LHX Slice in the Requirements Hyperspace, 1983 Settings

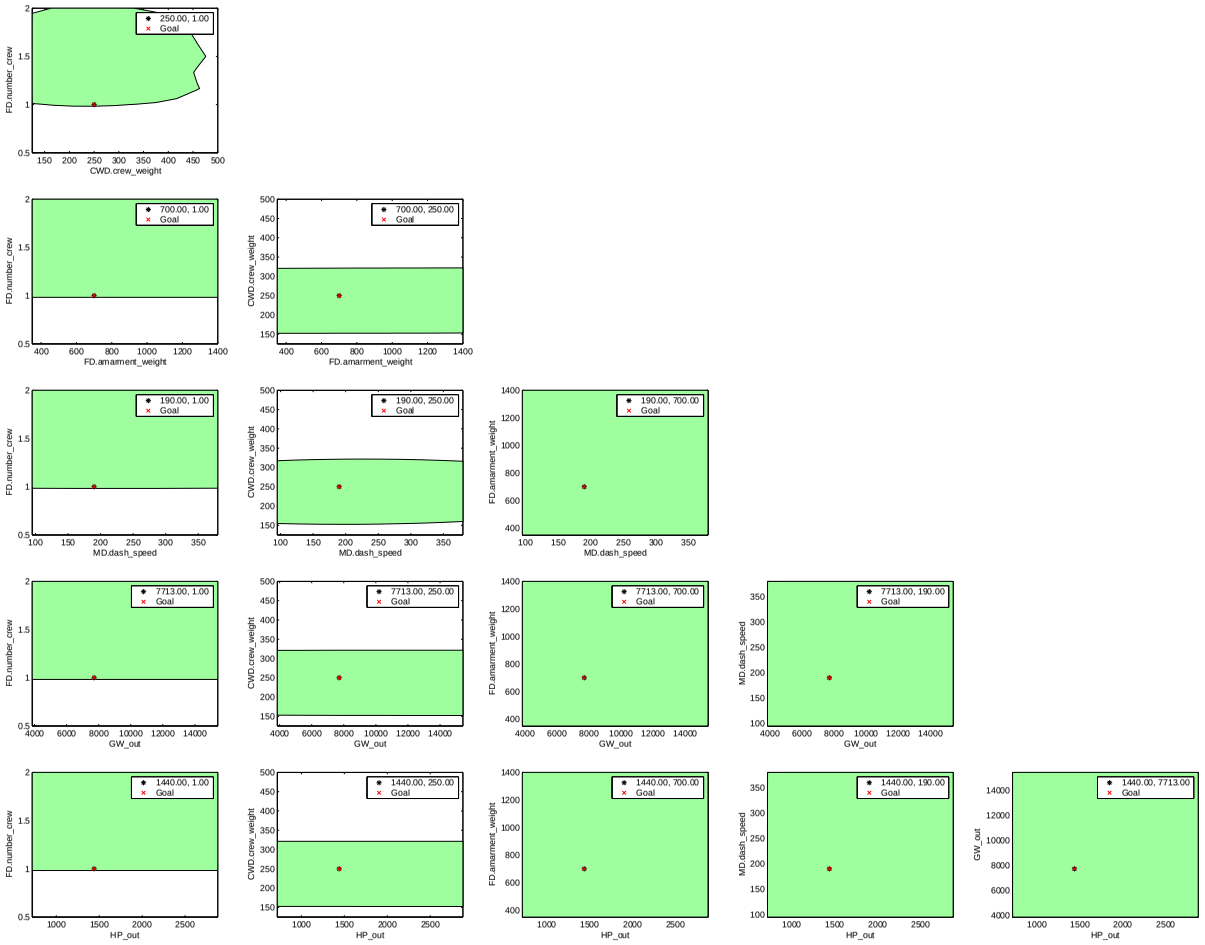


Figure 106: TDM Requirements Feasibility Region Sensitivity, 1983 Settings

G.2.3 Tilt-rotor Vehicle Type

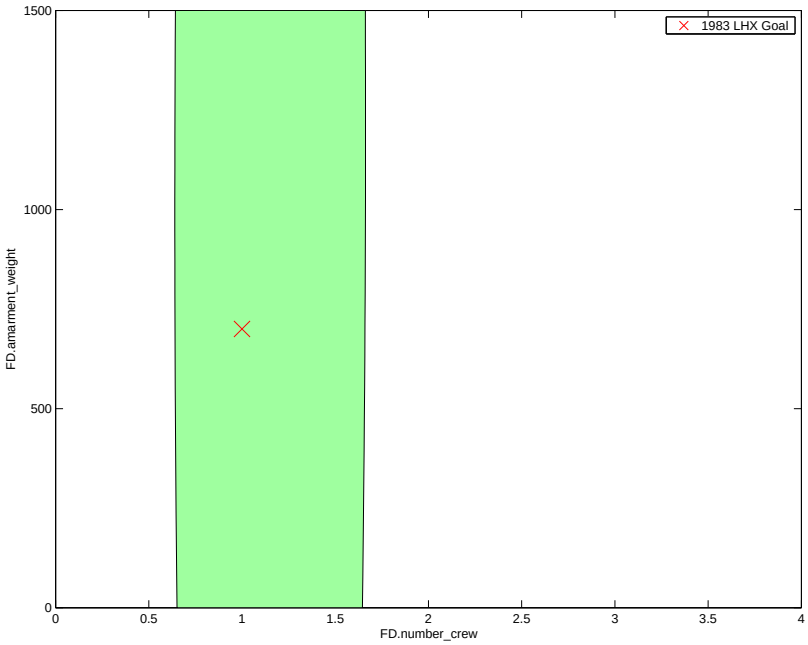
The fourth system, which is also one that was seriously considered during the early stages of the LHX program, is the tiltrotor. It was, therefore, of particular interest to determine whether or not the results from the RCD methodology in general and the MSPEA tool in particular matched the results from the LHX program.

The initial investigation of the feasible requirements (hyper)space at the original 1983 LHX requirements, shown in Figure 107, show the vehicle type to possesses a feasible space. This is not particularly surprising, since the three other vehicles were capable of meeting the original LHX requirements.

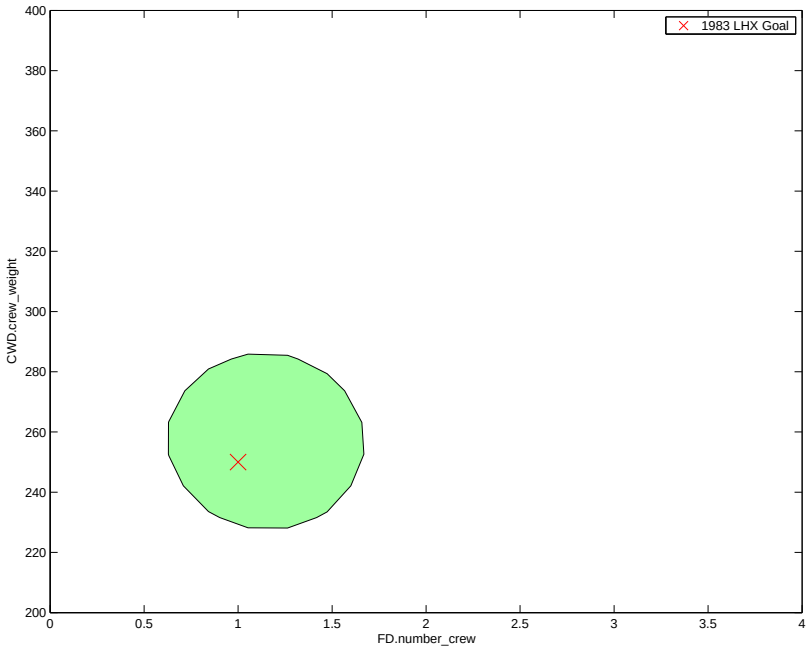
Because of the fact that the tilt-rotor does not rely on the rotor disk to provide both lift and propulsion during forward flight, it is possible for the maximum velocity of the tiltrotor to be significantly higher than that for a standard COAX, SMR, or TDM vehicle type. Therefore, the *MD.dash_speed* and *WD.dive_speed* requirements were adjusted to a higher velocity, 270 knots [116, pp. 159]. All of the other system level requirements were held constant. The resulting plots are shown in Figure 108. The resulting feasible regions in the requirements (hyper)space are slightly different. In fact the size of the regions increases as the dash-speed increases. This stems from the fact that the wing becomes more efficient as the speed is increased. A similar response would be seen if the cruise speeds were increased.

It is useful to look at the sensitivities of the TLTR vehicle system to the same requirements that all of the other vehicle types have been compared to, i.e. those in Figure 51. These are shown in Figure 109. Again, since the dash speed can be significantly higher for the TLTR than the pure rotorcraft; it is, therefore, of interest to see the difference in the sensitivities as the higher dash speed. These are shown in Figure 110.

Again, there isn't any information that is overly surprising. The TLTR vehicle type seems to be effected by the technology boundaries in a different manner from

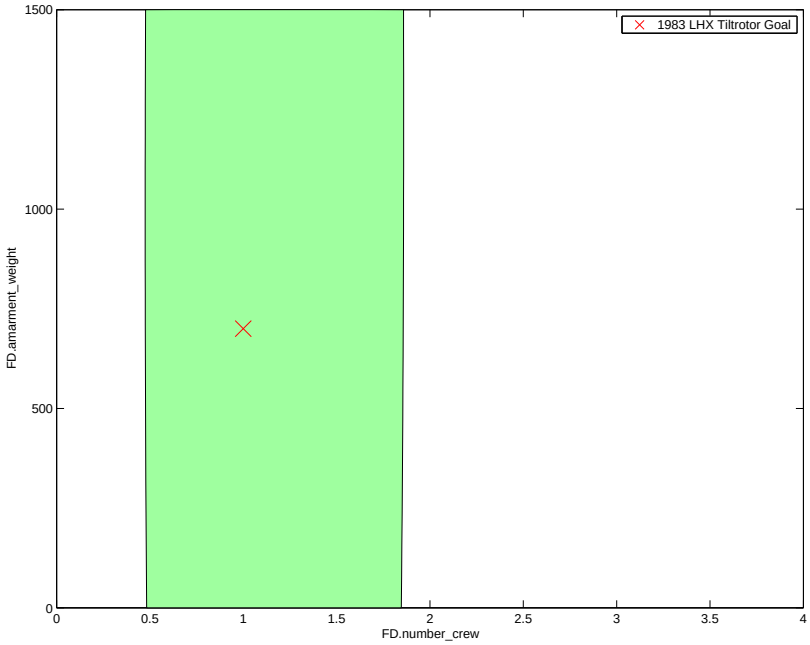


(a) Number of Crew vs. Armament Weight

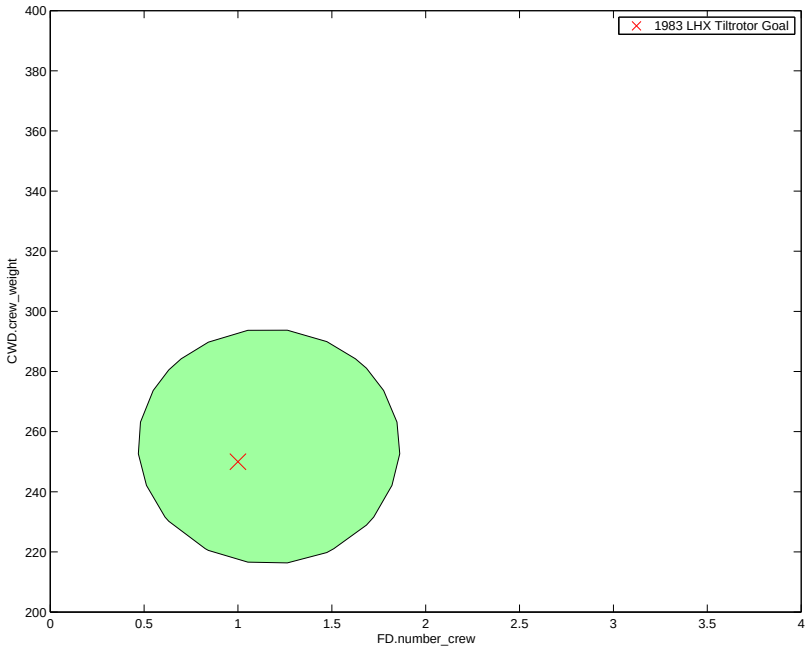


(b) Number of Crew vs. Crew Member Weight

Figure 107: TLTR LHX Slice in the Requirements Hyperspace, 1983 Requirements



(a) Number of Crew vs. Armament Weight



(b) Number of Crew vs. Crew Member Weight

Figure 108: TLTR LHX Slice in the Requirements Hyperspace, Adjusted Dash Speed

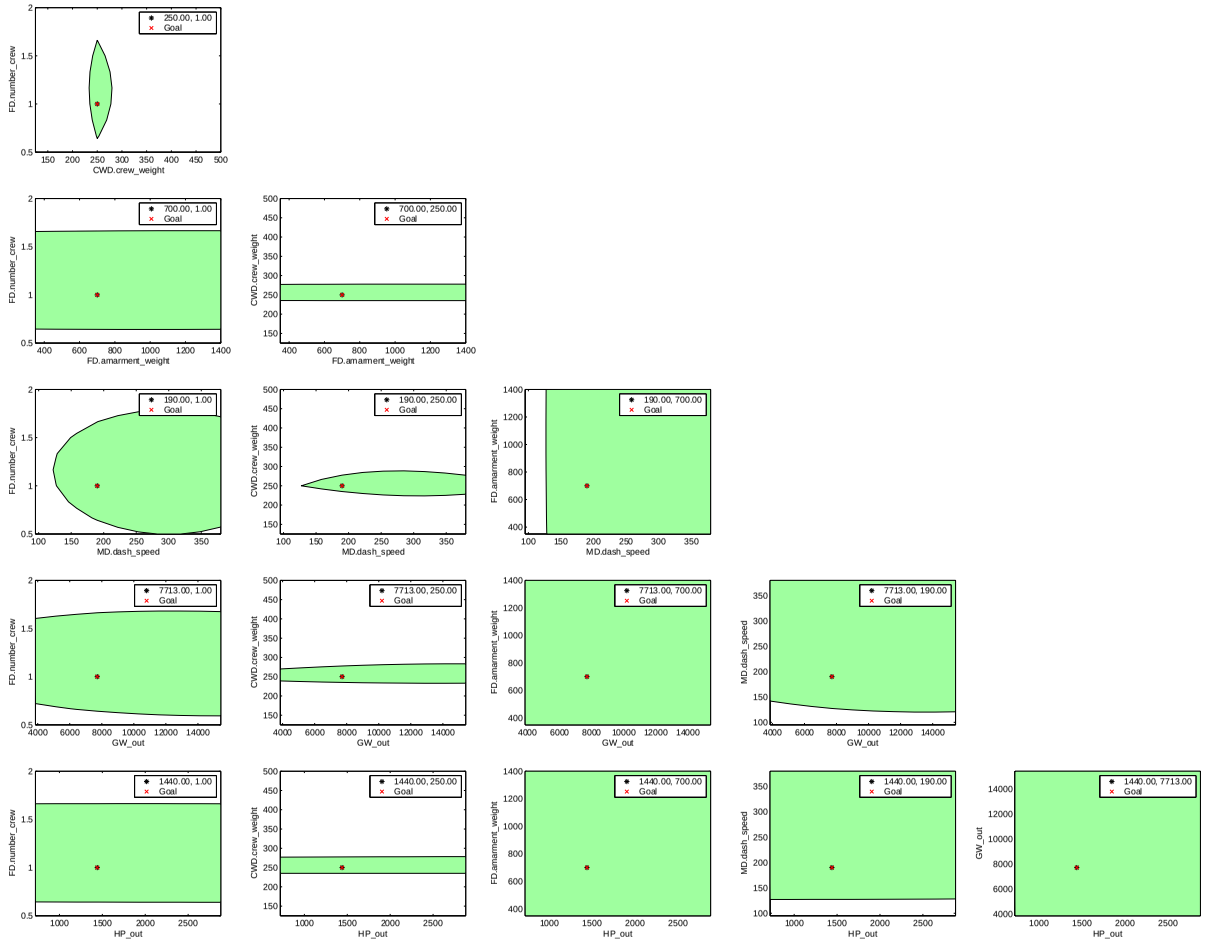


Figure 109: TLTR Requirements Feasibility Region Sensitivity

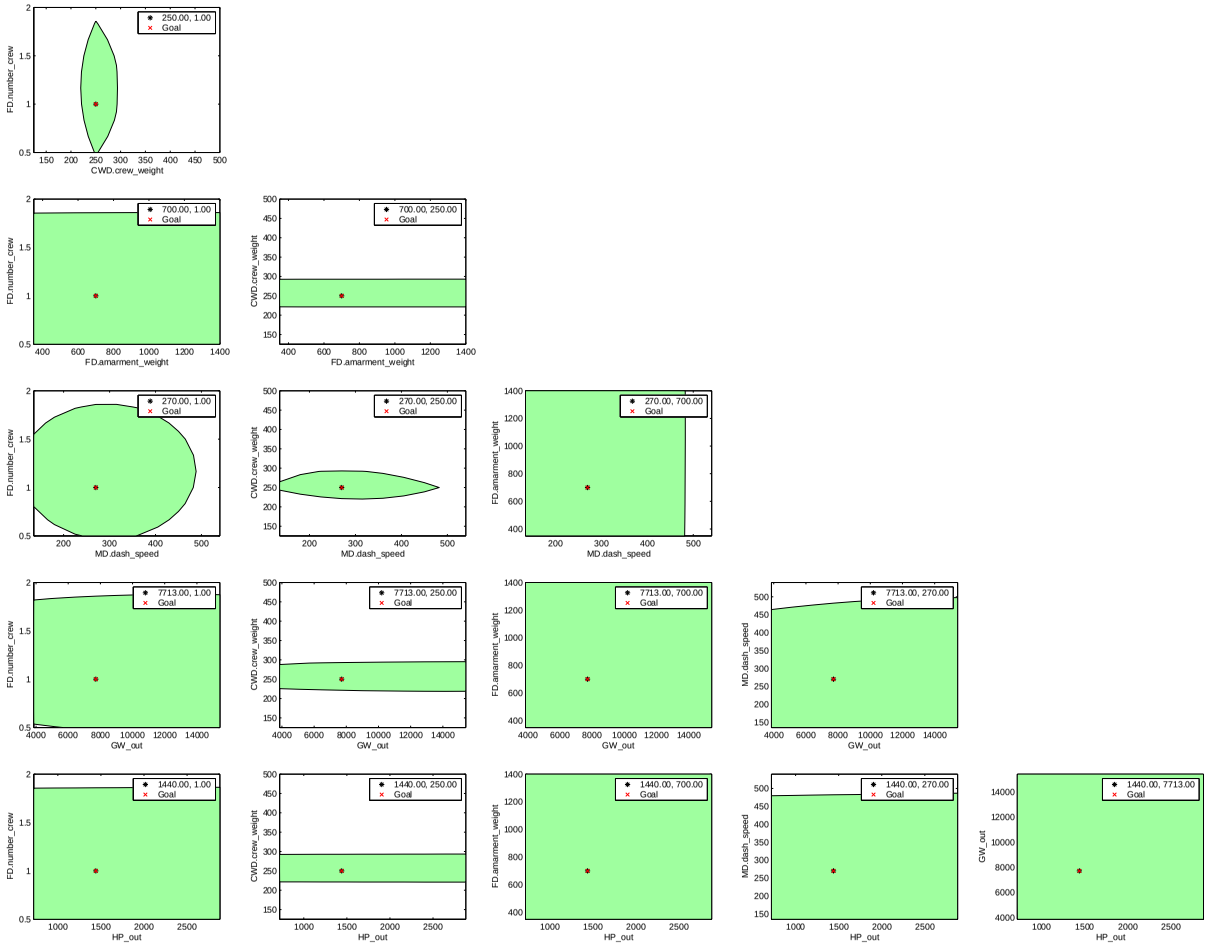


Figure 110: TLTR Requirements Feasibility Region Sensitivity

Table 29: Non-mission & Payload Requirements Adjustments for the Utility Mission [116, pp. 123]

Requirement	Combat Mission	Utility Mission
GW_out	7,713	9,124

the more conventional rotorcraft. This stems from the fact that the dynamics of the TLTR mission are different from those of more standard rotorcraft configurations, i.e. during a large portion of the mission profile the TLTR more closely resembles a fixed-wing aircraft as compared to a rotorcraft. Additionally, the TLTR has a significantly higher dash speed capability then any of the other vehicle types.

G.2.4 1983 LHX Utility/Transport Helicopters

Since the original LHX requirements also specified a transport/utility version of the helicopter, it is of interest to investigate the feasible regions of the requirements space for the transport version's requirements. The primary adjustments in requirements that needed to be made were in the area of payload and mission profile. Theses are shown in Table 8. Additionally, it was necessary to adjust the gross weight limit. The revised numbers are given in Table 29. This utility mission example was performed only upon the SMR vehicle type. While any or all of the other vehicle types could have been analyzed; this was not done since the LHX program ultimately dropped plans for the utility helicopter.

Since a utility mission calls for the ability to carry either passengers or payload it is useful to understand the trade-off made between the two. This is shown in Figure 111.

The mapping between payload and passengers is not quite so straightforward. This stems from the fact that the weight of each passenger is also a control variable. The relationship between the number of passengers and passenger weight is shown in Figure 112. The shape of this requirements space is also the effect of the "absolute" nature of the requirements settings and the technology boundaries. A similar

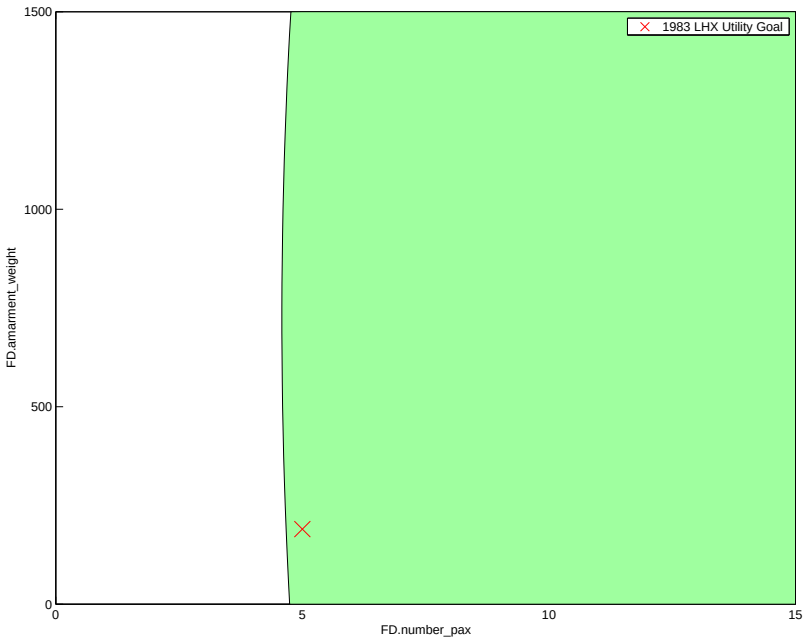


Figure 111: LHX Utility Mission, Number of Passengers vs. Other Payload Weight

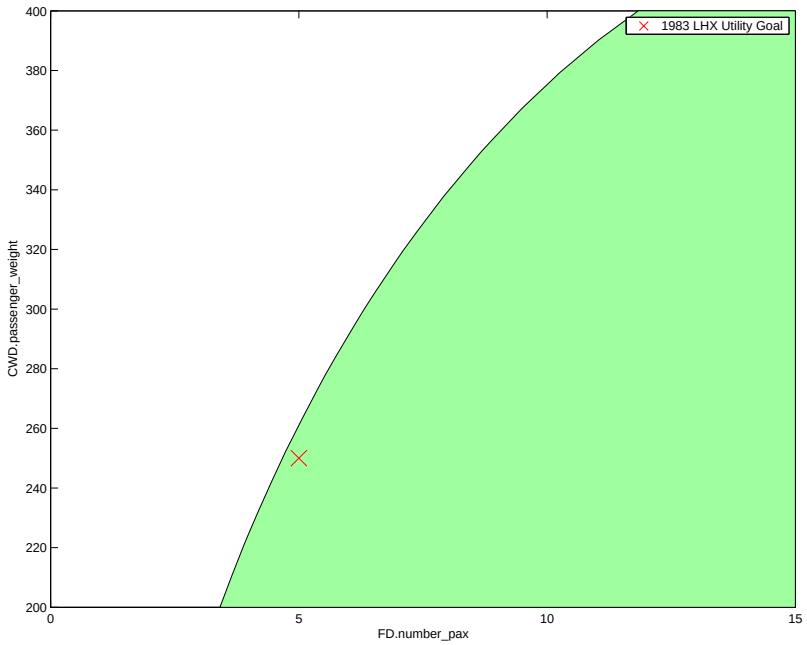


Figure 112: LHX Utility Mission, Number of Passengers vs. Passenger Weight

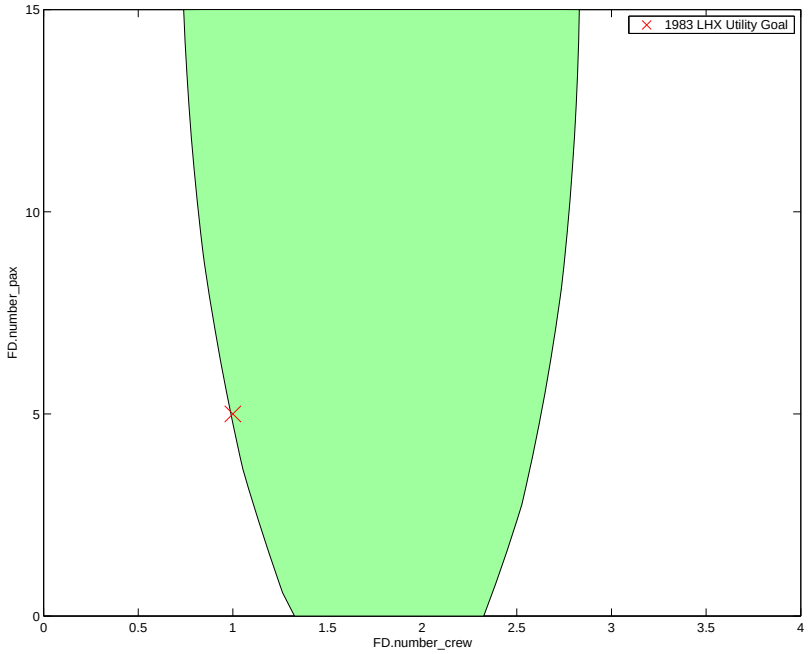


Figure 113: LHX Utility Mission, Number of Crew vs. Number of Passengers

relationship can be seen between the number of crew members and the number of passengers for a fixed secondary payload weight. This is shown in Figure 113.

The sensitivity relationships between the number of crew, passengers, and the size of the secondary payload with gross weight and horsepower are shown in Figure 114

G.3 RAH-66 Comanche Results

The results for the investigation of the COAX, TDM, and TLTR vehicle types for the RAH-66 Comanche requirements and technology boundaries are given below. Comparing the results to those shown in Section 6.3.4 to those presented below allows the reader to understand the different behaviors of the different vehicle types in greater detail.

G.3.1 Current Technology, COAX Vehicle System

The COAX vehicle type also proved capable of meeting the RAH-66 Comanche's requirements with the current technology levels. Two examples of this feasible space

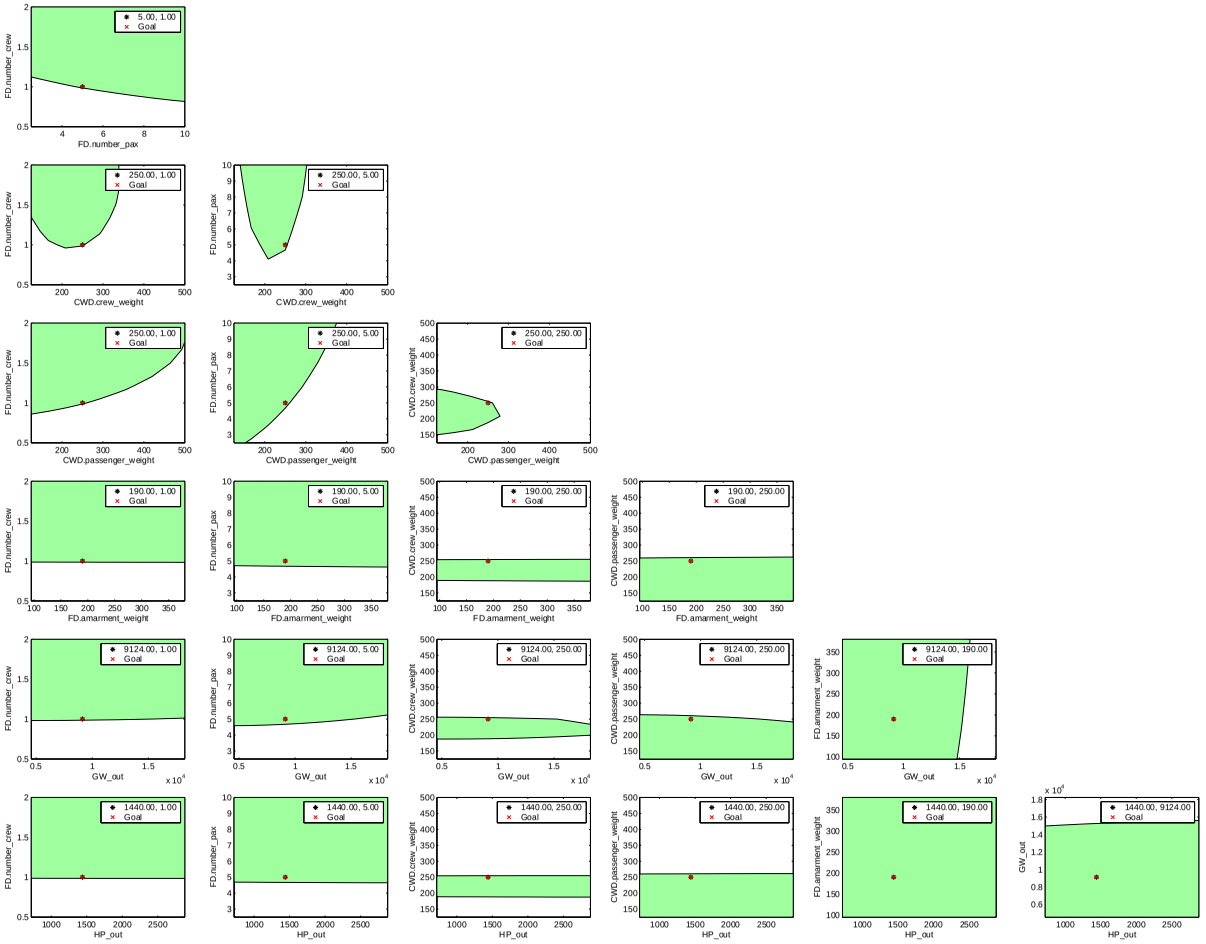


Figure 114: LHX Utility Mission, Payload Weight vs. Gross Weight and Horsepower Sensitivities

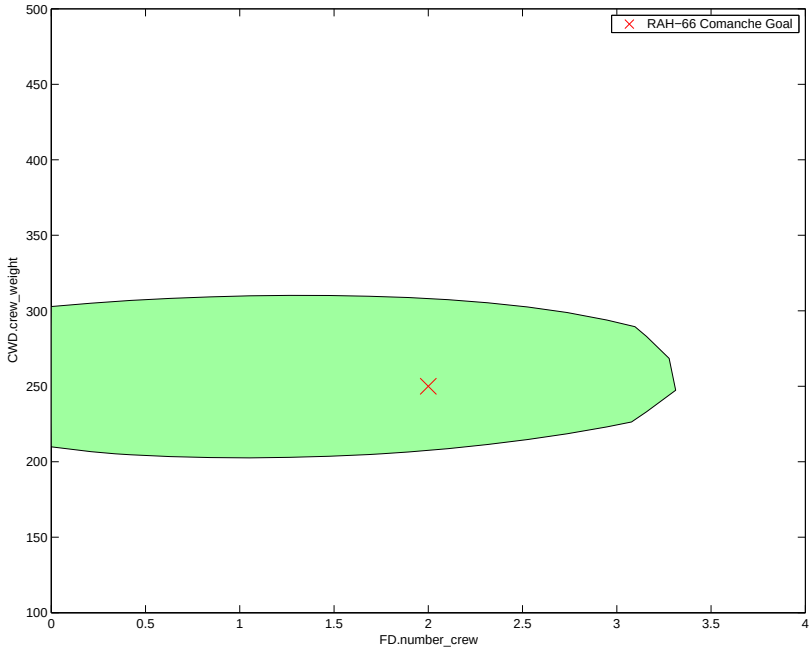


Figure 115: COAX Vehicle, Current Technologies, RAH-66 Requirements, Number of Crew vs Crew Member Weight

are given in Figures 115 and 116. It should be noted that the axes shown in Figure 60, armament weight vs. horsepower, produce a feasible region for its entire range for all of the vehicle types. Therefore, no additional knowledge can be obtained by displaying the same plot for the COAX or other vehicle systems.

Comparing the feasible space shown in Figure 116 with that for the SMR vehicle type, shown in Figure 62, it is clear that the shape of the requirements space, at this particular slice is substantially different for the COAX vehicle as compared to the SMR vehicle. Since it is not possible to compare the exact vehicles that make-up the boundary at this point, without further investigation, the engineer cannot know definitively why this occurs, but it most likely stems from the different behavior of the COAX vehicle that is inherent to the $\mathbf{R_f}$ tool. To get a further feel for this different behavior it is instructive to view multiple requirements slices simultaneously. This is shown in the now familiar sensitivity chart, Figure 117.

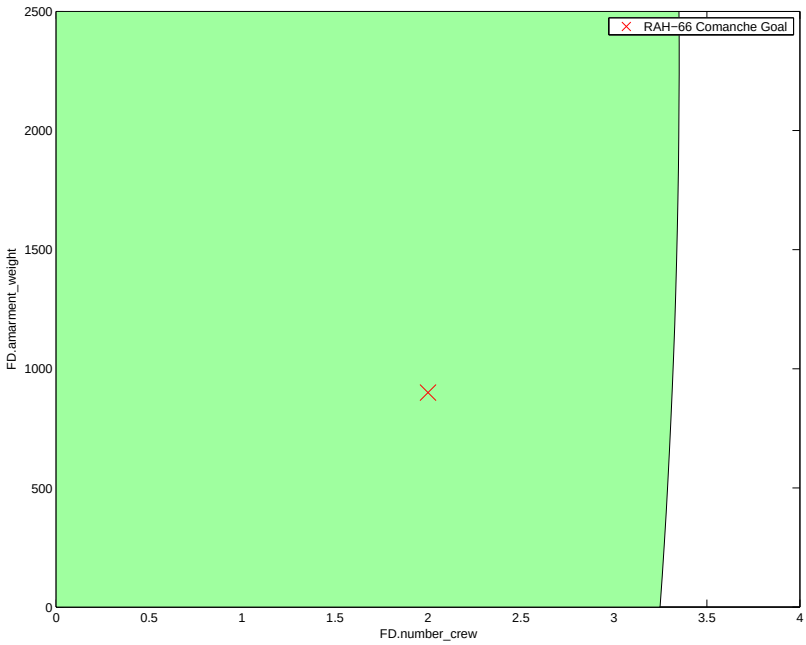


Figure 116: COAX Vehicle, Current Technologies, RAH-66 Requirements, Number of Crew vs Armament Weight

G.3.2 Current Technology, TDM Vehicle System

The TDM vehicle also showed improvement in the size of the feasible space when modified to incorporate the RAH-66 requirements and present day technology levels. Examples of the feasible requirements space are shown in Figures 118 and 119. Again the TDM vehicle type displays a different behavior than the previous two systems. Primarily because of the systems is way the $\mathbf{R_f}$ tool models different systems. What this implies is that, for the $\mathbf{R_f}$ tool, the system type definition, i.e. SMR, COAX, TDM, or TLTR, is actually a control variable that helps describe the behavior of the $\mathbf{R_f}$ tool. Since the primary purpose of the $\mathbf{R_f}$ tool is to model and size rotor-craft vehicles, by implication the vehicle type is a control variable for the behavior of the rotor-craft system type.

This poses a conundrum, since the control variables are inherently independent, meaning that no other variable is able to determine their setting, and the requirements are control variables; what does this imply for the system type, which is dependent

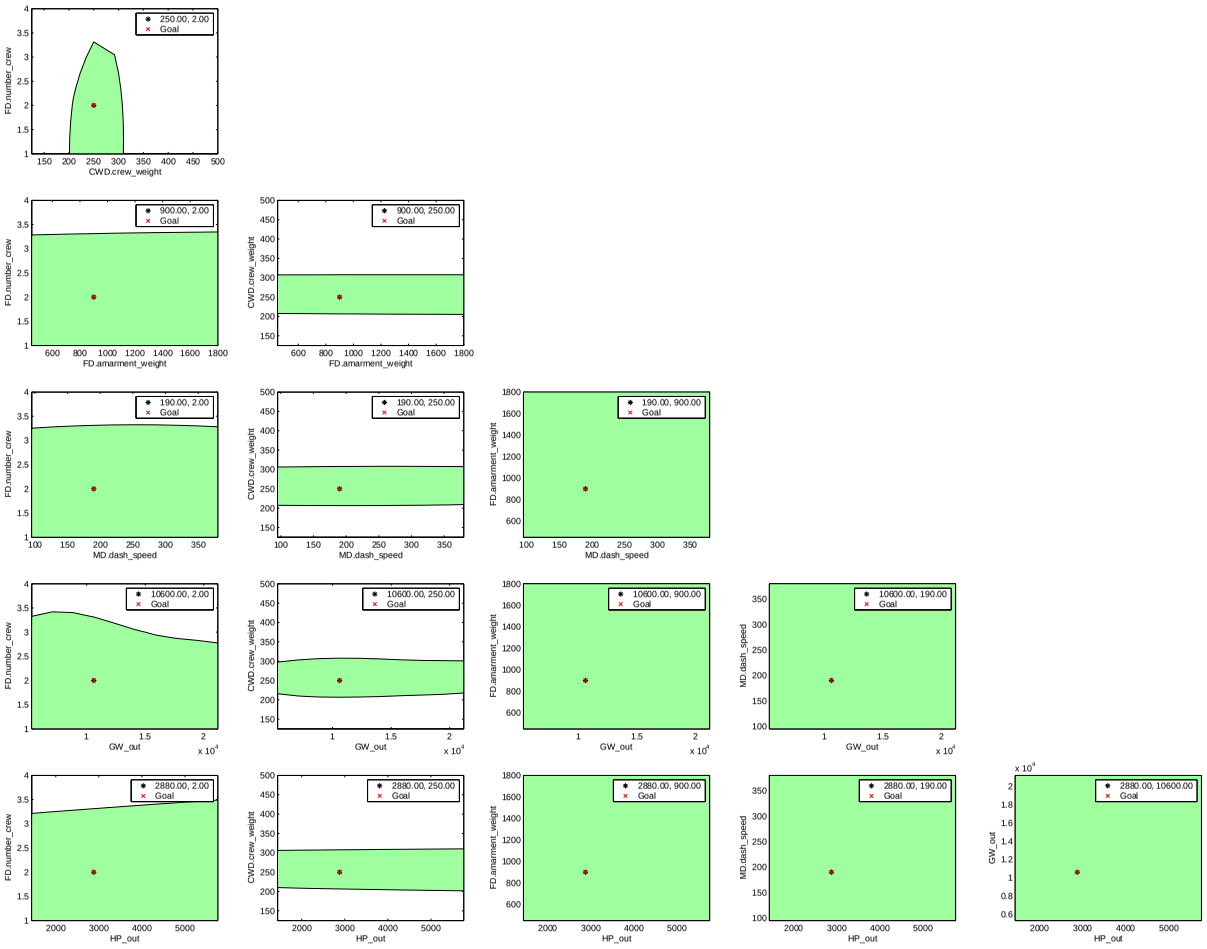


Figure 117: COAX Requirements Sensitivity, Current Technologies, RAH-66 Requirements

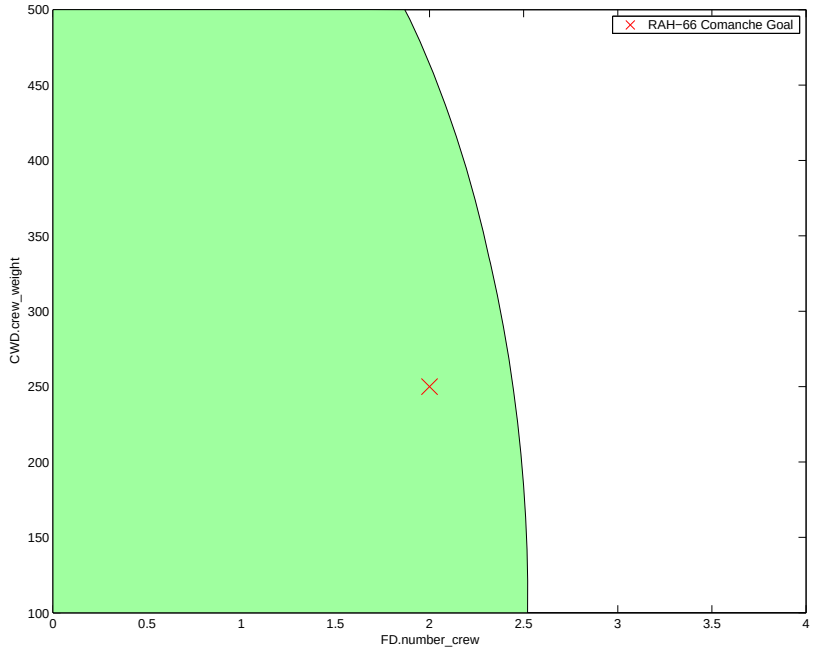


Figure 118: TDM Vehicle, Current Technologies, RAH-66 Requirements, Number of Crew vs Crew Member Weight

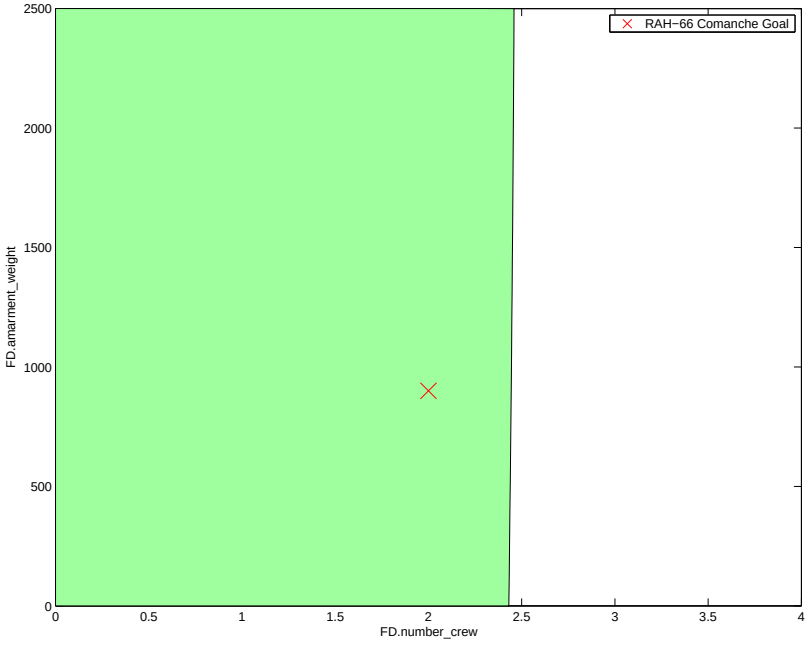


Figure 119: TDM Vehicle, Current Technologies, RAH-66 Requirements, Number of Crew vs Armament Weight

upon the settings of the other requirements. It is known that CVs can also be SVs, that is the independent CVs may actually help define the final state/response of the system. However, can there be such thing as a dependent CV? The system type, unless explicitly specified by the customer, is determined by the combination of the system requirements and the technology boundaries, but the system type also effects the behavior of the system. Definition 3 defines a CV as a variable which determines the behavior of the system, this implies that the system type is a control variable. However, the mathematical basis of catastrophe theory, described in Appendix A, states that CVs are independent variables. It is, therefore, necessary to rectify this inconsistency. The most straightforward way to do this, is to remember that unless the customer actually specifies that the system be a specific type of rotor-craft, the vehicle type only sets a "sub-behavior" of the total rotor-craft "behavior". That is it is just like many of the other state variables it defines the behavior of one or more subsystems or components. Therefore, under most circumstances the vehicle type is actually a state variable for the rotor-craft system type.

In comparing the TDM to the other vehicle types it is, as usual, beneficial to investigate the sensitivity of the feasible space to multiple combinations of requirements. This is shown in Figure 120.

G.3.3 Current Technology, TLTR Vehicle System

The TLTR system, which is the most dissimilar of all of the vehicle types investigated is also capable of meeting the basic RAH-66 Comanche's requirements. Examples of this are given in Figures 121 and 122. The sensitivities of the feasible requirements space to multiple combinations of these requirements is again shown in Figure 123.

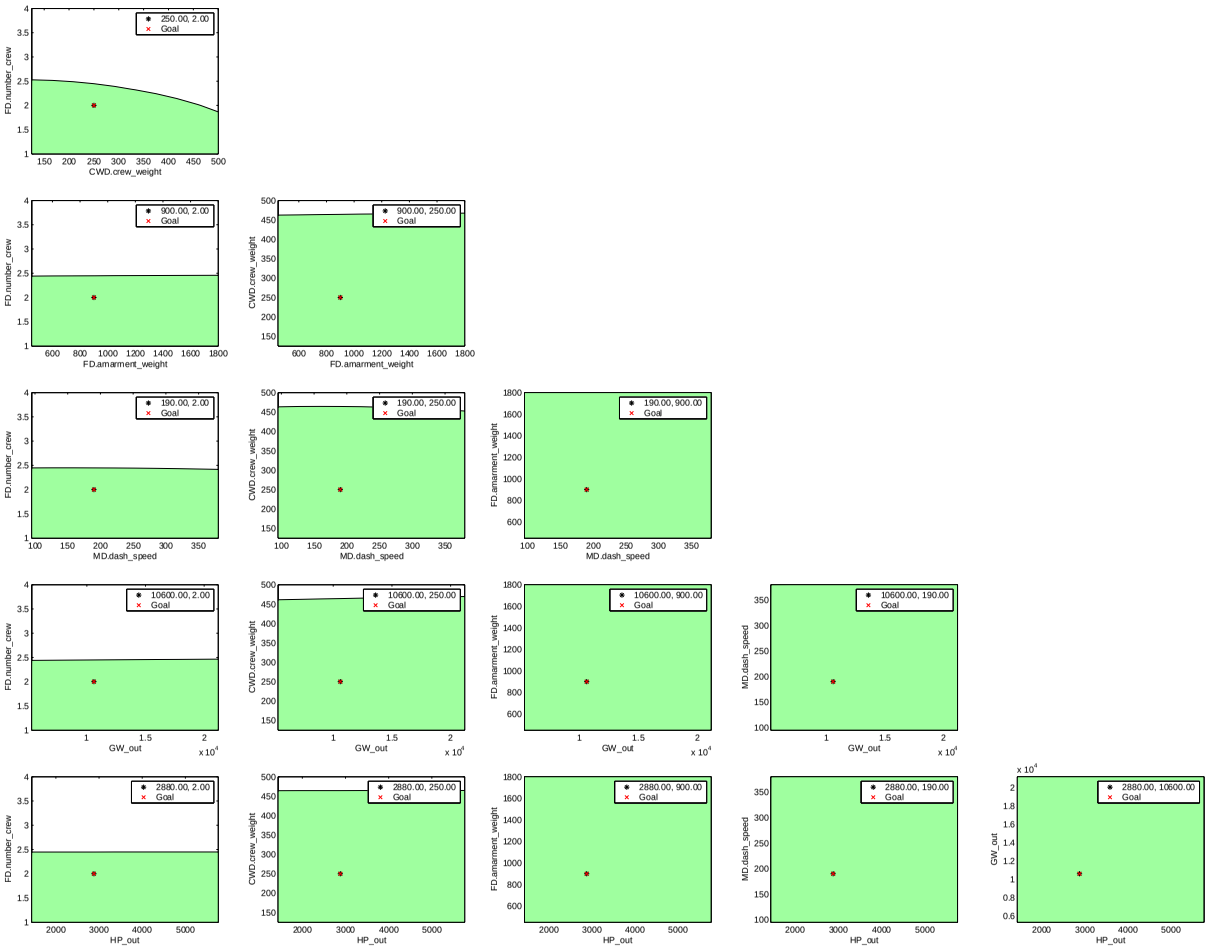


Figure 120: TDM Requirements Sensitivity, Current Technologies, RAH-66 Requirements

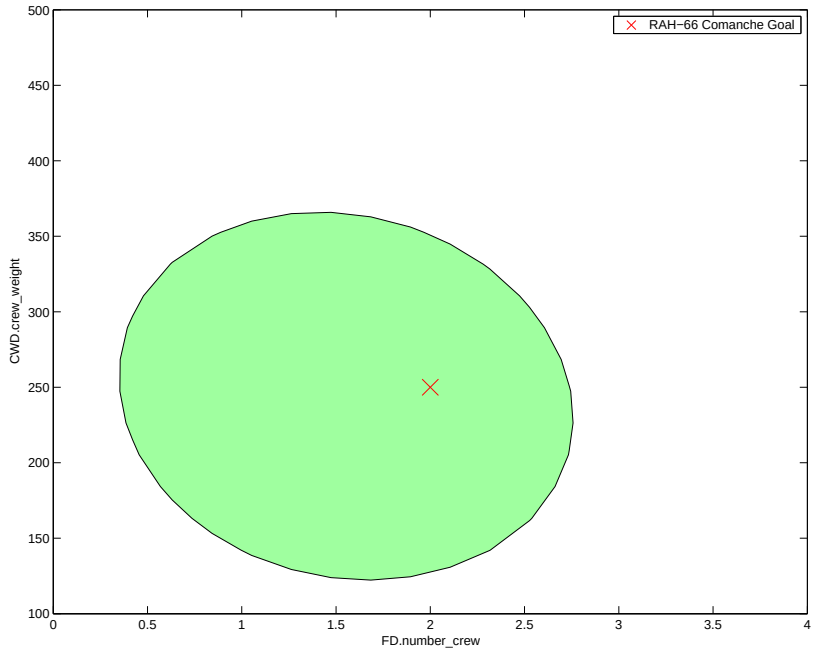


Figure 121: TLTR Vehicle, Current Technologies, RAH-66 Requirements, Number of Crew vs Crew Member Weight

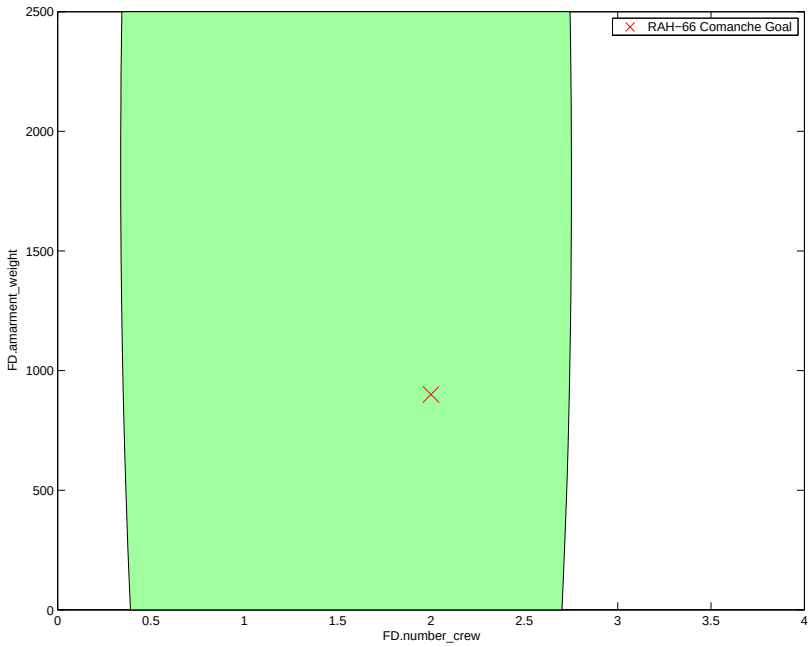


Figure 122: TLTR Vehicle, Current Technologies, RAH-66 Requirements, Number of Crew vs Armament Weight

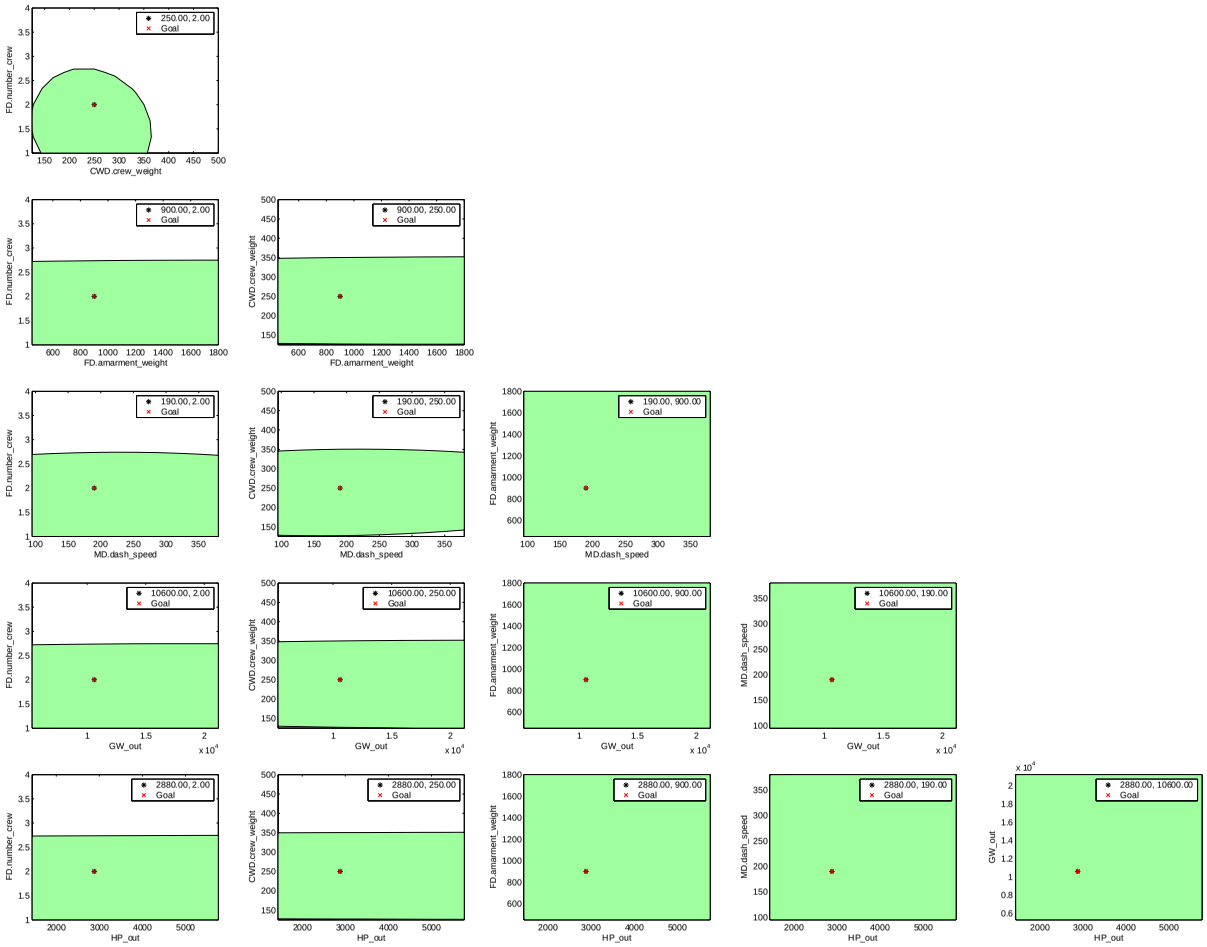


Figure 123: Multiple Vehicle Type Feasible Requirements Regions, Current Technologies, RAH-66 Requirements

REFERENCES

[1] Mavris, D. N. and DeLaurentis, D., "Methodology for Examining the Simultaneous Impact of Requirements, Vehicle Characteristics, and Technologies on Military Aircraft Design," in *22nd Congress of the International Council on the Aeronautical Sciences (ICAS)*, (Harrogate, England), August 27-31 2000. Available from: `http://www.asdl.gatech.edu/publications/pdf/2000/ICAS-2000-144.pdf`.

[2] Baker, A. P., *The Role of Mission Requirements, Vehicle Attributes, Technologies and Uncertainty in Rotorcraft System Design*. PhD thesis, Georgia Institute of Technology, March 2002.

[3] Mavris, D. N., Baker, A. P., and Schrage, D. P., "Simultaneous Assessment of Requirements and Technologies in Rotorcraft Design," in *Annual Forum of the American Helicopter Society*, (Virginia Beach, VA), May 2-4 2000. Available from: `http://www.asdl.gatech.edu/publications/pdf/2000/AHS-56-AD1-1.pdf`.

[4] Baker, A. P. and Mavris, D. N., "Assessing the Simultaneous Impacts of Requirements, Vehicle Characteristics, and Technologies During Aircraft Design," in *39th AIAA, Aerospace Sciences Meeting and Exhibit*, (Reno, NV), January 9-11 2001. Available from: `http://www.asdl.gatech.edu/publications/pdf/2001/AIAA-2001-0533.pdf`.

[5] Baker, A. P., Mavris, D. N., and Schrage, D. P., "Assessing the Impact of Mission Requirements, Vehicle Attributes, Technologies and Uncertainty in Rotorcraft System Design," in *58th Annual Forum*, (Montreal, Canada), American Helicopter Society, June 11-13 2002. Available from: `http://www.asdl.gatech.edu/publications/pdf/2002/AHS-2002-B.pdf`.

[6] Hollingsworth, P. and Mavris, D. N., "A Method for Concept Exploration of Hypersonic Vehicle in the Presence of Open & Evolving Requirements," in *5th World Aviation Congress and Exposition*, (San Diego, CA), Society of Automotive Engineers, American Institute of Aeronautics and Astronautics, October 2000. Available from: `http://www.asdl.gatech.edu/publications/pdf/2000/AIAA-2000-01-5560.pdf`.

[7] Hollingsworth, P. and Mavris, D., "Identification of the Requirements Space Topology for a Rapid Response Strike System," *SAE 2001 Transactions: Journal of Aerospace*, vol. 110, pp. 663–673, Sec. 1 2002. Available from: `http://www.asdl.gatech.edu/publications/pdf/2001/SAE-2001-01-3017.pdf`.

[8] Schrage, D. P. and Mavris, D. N., "Integrated Product/Process Design/Development (IPPD) Through Robust Design Simulation," in *1st AIAA Aircraft Engineering, Technology, and Operations Congress*, (Los Angeles, CA), September 1995.

[9] Fabrycky, W. and Blanchard, B., *Life-Cycle Cost and Economic Analysis*. Englewood Cliffs: Prentice-Hall, 1991.

[10] Kirby, M. R., *A Methodology for Technology Identification, Evaluation, and Selection in Conceptual and Preliminary Aircraft Design*. PhD thesis, Georgia Institute of Technology, 2001.

[11] Mavris, D. N., Baker, A., and Schrage, D. P., "IPPD Through Robust Design Simulation for an Affordable Short Haul Civil Tiltrotor," in *American Helicopter Society 53rd Annual Forum*, (Virginia Beach, VA), American Helicopter Society, Inc., April-May 1997.

[12] "DoD Guide to Integrated Product and Process Development," Tech. Rep. Version 1.0, Office of the Under Secretary of Defense, Department of Defense, February 5, 1996.

[13] Marx, W. J., Mavris, D. N., and Schrage, D. P., "Integrating Design and Manufacturing for the High Speed Civil Transport," American Institute of Aeronautics and Astronautics, Inc., September 1994. Available from: `http://www.asdl.gatech.edu/publications/pdf/1994/ICAS-94-1084.pdf`.

[14] Schrage, D. P. and Mavris, D. N., "Recomposition: The Other Half of the MDO Equation," in *INCASE/NASA Langley Workshop in Multi-Disciplinary Design Optimization-State-of-the-Art*, (Langley, VA), 1995.

[15] Mavris, D. N. and Bandte, O., "Economic Uncertainty Assessment Using a Combined Design of Experiments/Monte Carlo Simulation Approach with Application to an HSCT," May 1995. Available from: `http://www.asdl.gatech.edu/publications/pdf/1995/ISPA-95.pdf`.

[16] Mavris, D. N., DeLaurentis, D. A., Bandte, O., and Hale, M. A., "A Stochastic Approach to Multi-disciplinary Aircraft Analysis and Design," in *36th Aerospace Sciences Meeting and Exhibit*, (Reno, NV), 1998. Available from: `http://www.asdl.gatech.edu/publications/pdf/1998/AIAA-98-0912.pdf`.

[17] Mavris, D. N., DeLaurentis, D. A., Hale, M. A., and Tai, J. C., "Elements of an Emerging Virtual Stochastic life Cycle Design Environment," in *4th World Aviation Congress and Exposition*, (San Francisco, CA), AIAA, October 19-21 1999. Available from: `http://www.asdl.gatech.edu/publications/pdf/1999/AIAA-99-01-5638.pdf`.

[18] Mavris, D. N., Mantis, G., and Kirby, M. R., "Demonstration of a Probabilistic Technique for the Determination of Economic Viability," in *2nd World Aviation Congress and Exposition*, (Anaheim, CA), October 1997.

[19] Mavris, D. N. and DeLaurentis, D. A., "A Stochastic Design Approach for Aircraft Affordability," in *1st Congress of the International Council on the Aeronautical Sciences (ICAS)*, (Melbourne, Australia), September 1998.

[20] Kirby, M. R. and Mavris, D. N., "Forecasting the Impact of Technology Infusion on Subsonic Transport Affordability," in *Presented at the 3rd World Aviation Congress and Exposition*, (Anaheim, CA), September 28-30 1998. Available from: `http://www.asdl.gatech.edu/publications/pdf/1998/AIAA-98-5576.pdf`.

[21] *FPI User's and Theoretical Manual*. San Antonio, TX, 1995.

[22] Bandte, O., *A Probabilistic Multi-Criteria Decision Making Technique for Conceptual and Preliminary Aerospace Systems Design*. PhD thesis, Georgia Institute of Technology, September 2000.

[23] Mavris, D. N. and Bandte, O., "A Probabilistic Approach to Multivariate Constrained Robust Design Simulation," October 1997.

[24] Simpson, T. W., *A Concept Exploration Method for Product Family Design*. PhD thesis, Georgia Institute of Technology, 1998.

[25] Daberkow, D. D., *A Formulation of Metamodel Implementation Processes for Complex Systems Design*. PhD thesis, Georgia Institute of Technology, June 2002.

[26] Martin, J. D. and Simpson, T. W., "Use of Adaptive Metamodeling for Design Optimization," in *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, (Atlanta, GA), September 4-6 2002.

[27] Putter, H. and Young, G. A., "On the effect of covariance function estimation on the accuracy of kriging predictions." 1991.

[28] Scharl, J., *Formulation and Implementation of a Methodology for Dynamic Modeling and Simulation in Early Aerospace Design*. PhD thesis, Georgia Institute of Technology, November 2001.

[29] MacKay, D. J. C., "Gaussian Processes: A Replacement for Supervised Neural Networks?," tech. rep., Cavendish Laboratory, Cambridge University, Cambridge, CB3 OHE, UK. Available from: `http://www.inference.phy.cam.ac.uk/mackay/`.

[30] Storkey, A. J., *Efficient Covariance Matrix Methods for Bayesian Gaussian Processes and Hopfield Neural Networks*. PhD thesis, Imperial College, University of London, London, England, January 1999.

[31] Liong, S.-Y., Khu, S.-T., and Chan, W.-T., "Derivation of Pareto Front with Genetic Algorithm and Neural Network," *Journal of Hydrologic Engineering*, vol. 6, pp. 52–61, January/February 2001.

[32] Daberkow, D. D., "An Investigation of Metamodeling Techniques for Complex Systems Design," in *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, (Atlanta, GA), September 2002.

[33] MacKay, D. J. C., "Introduction to Gaussian Processes," paper, Department of Physics, Cambridge University, Cambridge, England, UK, 1998. Available from: `ftp://www.inference.phy.cam.ac.uk/pub/mackay/gpB.ps.gz`.

[34] Rasmussen, C. E., *Evaluation of Gaussian Processes and other Methods for Non-Linear Regression*. PhD thesis, Department of Computer Science, University of Toronto, Toronto, Ontario Canada, 1996.

[35] Bailer-Jones, C. A. L., "A Summary of Gaussian Processes," white paper, Cavendish Laboratory, University of Cambridge, Cambridge, England, UK.

[36] Gibbs, M. N., *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, University of Cambridge, Cambridge, England UK, 1997.

[37] Gibbs, M. and MacKay, D. J. C., "Efficient Implementation of Gaussian Processes," paper, Cavendish Laboratory, Cambridge, England, UK, May 1997. Available from: `http://www.cs.toronto.edu/~mackay/gpros.ps.gz`.

[38] Williams, C. K. I. and Barber, D., "Bayesian Classification with Gaussian Processes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 1342–1351, December 1998.

[39] Neal, R. M., "Monte Carlo Implementation of Gaussian Process Models for Bayesian Regression and Classification," Tech. Rep. 9702, Department of Statistics, University of Toronto, Toronto, Ontario, Canada, 1997.

[40] Csató, L. and Opper, M., "Sparse Online Gaussian Processes," Tech. Rep. NCRG/2001/01, Neural Computing Research Group, Aston University, Birmingham, B4 7ET, UK, 2001. Available from: `http://www.ncrg.aston.ac.uk`.

[41] Hariz, S. B., "Limit theorems for the Non-linear Functional of Stationary Gaussian Processes," *Journal of Multivariate Analysis*, 2001. Available from: `http://www.idealibrary.com`.

[42] Akcoglu, M. A., Baxter, J. R., Ha, D. M., and Jones, R. L., "Approximation of the L_2-Processes by Gaussian Processes," *New York Journal of Mathematics*, no. 4, pp. 75–82, 1998.

[43] Malzahn, D. and Opper, M., "Learning Curves for Gaussian Processes Regression: A Framework for Good Approximations," tech. rep., Neural Computing Research Group, Aston University, Birmingham, B4 7ET, UK. Available from: `http://www.ncrg.aston.ac.uk`.

[44] Csató, L. and Opper, M., "Sparse Representation for Gaussian Process Models," in *Advances in Neural Information Processing Systems* (Leen, T., Diettrich, T. G., and Tresp, V., eds.), MIT Press, 13 ed., 2001.

[45] Opper, M. and Vivarelli, F., "General Bounds on Bayes Errors for Regression with Gaussian Processes," tech. rep., Neural Computing Research Group, Aston University, Birmingham, B4 7ET, UK. Available from: `http://www.ncrg.aston.ac.uk`.

[46] Csató, L., Fokoué, E., Opper, M., Schottky, B., and Winther, O., "Efficient Approaches to Gaussian Process Classification," tech. rep., Neural Computing Research Group, Aston University, Birmingham, B4 7ET, UK. Available from: `http://www.ncrg.aston.ac.uk`.

[47] Cornford, D., Nabney, I. T., and Williams, C. K. I., "Modelling Frontal Discontinuities in Wind Fields," Tech. Rep. NCRG/99/001, Neural Computing Research Group, Aston University, Birmingham, B4 7ET, Uk, 1999. Available from: `http://www.ncrg/aston.ac.uk`.

[48] Bishop, C. M., Svensén, M., and Williams, C. K. I., "Development of the Generative Topographic Mapping," tech. rep., Microsoft Research Cambridge, Cambridge, CB2 3NH, UK, 1998.

[49] Williams, C. K. I., "Prediction with Gaussian Processes: From Linear Regression to Linear Prediction and Beyond," Tech. Rep. NCRG/97/012, Neural Computing Research Group, Aston University, Birmingham, B4 7ET, UK, 1997. Available from: `http://www.ncrg.aston.ac.uk`.

[50] Seeger, M., "Bayesian Model Selection for Support Vector Machines, Gaussian Processes and other Kernel Classifiers," tech. rep., Institute for Adaptive and Neural Computation, University of Edinburgh, Edinburgh Eh1 2QL, UK.

[51] Seeger, M., "Relationships between gaussian processes, support vector machines and smoothing splines." 2000.

[52] Mavris, D. N. and Kirby, M. R., "Technology Identification, Evaluation, and Selection for Commercial Transport Aircraft," in *Presented at the 58th Annual Conference Of Society of Allied Weight Engineers*, (San Jose, California), May 24-26 1999. Available from: `http://www.asdl.gatech.edu/publications/pdf/1999/SAWE-99-2456.pdf`.

[53] Kirby, M. R. and Mavris, D. N., "Forecasting Technology Uncertainty in Preliminary Aircraft Design," in *4th World Aviation Congress and Exposition*, (San Francisco, CA), October 19-21 1999. Available from: `http://www.asdl.gatech.edu/publications/pdf/1999/AIAA-99-01-5631.pdf`.

[54] Kirby, M. R. and Mavris, D. N., "A Technique for Selecting Emerging Technologies for a Fleet of Commercial Aircraft to Maximize R&D Investment,"

in *SAE Aerospace Congress and Exhibition*, Aerospace Systems Design Laboratory, Georgia Institute of Technology, Society of Automotive Engineers, September 2001. Available from: http://www.asdl.gatech.edu/publications/pdf/2001/SAE-2001-01-3018.pdf.

[55] Kirby, M. R., Mavris, D. N., and Largent, M. C., "A Process for Tracking and Assessing Emerging Technology Development Programs for Resource Allocation," in *1st AIAA ATIO Forum*, (Los Angeles, CA), October 16-18 2001. Available from: http://www.asdl.gatech.edu/publications/pdf/2001/AIAA-2001-5280.pdf.

[56] Mavris, D. N., Kirby, M. R., and Qui, S., "Technology Impact Forecasting for a High Speed Civil Transport," in *Presented at the 3rd World Aviation Congress and Exposition*, (Anaheim, CA), September 28-30 1998. Available from: http://www.asdl.gatech.edu/publications/pdf/1998/AIAA-98-5547.pdf.

[57] Roth, B. and Mavris, D., "Analysis of Advanced Technology Impact on HSCT Engine Cycle Performance," in *35th Joint Propulsion Conference*, (Los Angeles, CA), June 1999. Available from: http://www.asdl.gatech.edu/publications/pdf/1999/AIAA-99-2379.pdf.

[58] Mavris, D. N., Baker, A. P., and Schrage, D. P., "Implementation of a Technology Impact Forecast Technique on a Civil Tiltrotor," in *Proceedings of the 55th National Forum of the American Helicopter Society*, (Montreal, Quebec, Canada), May 25-27 1999. Available from: http://www.asdl.gatech.edu/publications/pdf/1999/AHS-99-AB.pdf.

[59] Mavris, D. N., Soban, D. S., and Largent, M. C., "An Application of a Technology Impact Forecasting (TIF) Method to an Uninhabited Combat Aerial Vehicle," in *4th World Aviation Congress and Exposition*, (San Francisco, CA), October 19-21 1999. Available from: http://www.asdl.gatech.edu/publications/pdf/1999/AIAA-99-01-5633.pdf.

[60] Roth, B., German, B., Mavris, D., and Macsotai, N., "Adaptive Selection of Engine Technology Solution Sets from a Large Combinatorial Space," July 2001. Available from: http://www.asdl.gatech.edu/publications/pdf/2001/AIAA-2001-3208.pdf.

[61] Stettner, M. and Schrage, D. P., "Tiltrotor Performance Sensitivities for Multidisciplinary Wing Optimization," in *Presented at the American Helicopter Society International Specialists Meeting on Rotorcraft Multidisciplinary Design Optimization*, (Atlanta, GA), April 27-28 1993. Available from: http://www.asdl.gatech.edu/publications/pdf/1993/AHS-0493-15.pdf.

[62] Mavris, D. N., Tai, J., and Schrage, D. P., "A Multidisciplinary Design Optimization Approach to Sizing Stopped Rotor Configurations Utilizing Reaction Drive and Circulation Control," in *Presented at the 5th AIAA/-NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, (Panama City, FL), September 7-9 1994. Available from: `http://www.asdl.gatech.edu/publications/pdf/1994/AIAA-94-4296.pdf`.

[63] Kroo, I., Altus, S., Bruan, R., Gage, P., and Sobieski, I., "Multidisciplinary Optimization Methods for Aircraft Preliminary Design," in *Fifth AIAA/USAF/-NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, (Panama City, Florida), September 1994. Available from: `http://aero.stanford.edu/Reports/MDO94.html`.

[64] Olds, J. R., "The Suitability if Selected Multidisciplinary Design and Optimization Techniques to Conceptual Aerospace Vehicle Design," in *Presented at 4th AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization*, (Cleveland, OH), Spetember 1992.

[65] Olds, J. R. and Ledsinger, L. A., "Multidisciplinary Optimization Techniques for Branching Trajectories," in *Presented at the AIAA 36th Aerospace Sciences Meeting and Exhibit*, (Reno, NV), January 1998. Available from: `http://www.ssdl.gatech.edu/main/ssdl_paper_archive/aiaa_98asm.pdf`.

[66] Braun, R., Gage, P., Kroo, I., and Sobieski, I., "Implementation and Performance Issues in Collaborative Optimization," in *Proceedings of the Sixth AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, (Bellevue, WA), September 1996. Available from: `http://citeseer.nj.nec.com/rd/66428856%2C3307%2C1%2C0.25%2CDownload/http://citeseer.nj.nec.com/cache/papers/cs/3502/ftp:zSzzSztechreports.larc.nasa.govzSzpubzSztechreportszSzlarczSz96zSzNASA-aiaa-96-4017.pdf/braun96implementation.pdf`.

[67] Braun, R. D. and Kroo, I. M., "Development and Application of the Collaborative Optimization Architecture in a Multidisciplinary Design Environment," in *Multidisciplinary Design Optimization: State-of-the-Art*, SIAM, 1997.

[68] Cormier, T. A., Scott, A., Ledsinger, L. A., McCormick, D. J., Way, D. W., and Olds, J. R., "Comparison of Collaborative Optimization to Conventional Design Techniques for a Conceptual RLV," in *th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, (Long Beach, CA), September 2000. Available from: `http://www.ssdl.gatech.edu/main/ssdl_paper_archive/AIAA-2000-4885.pdf`.

[69] Budianto, I. A. and Olds, J. R., "A Collaborative Optimization Approach to Design and Deployment of a Space Based Infrared System Constellation," in *2000 IEEE Aerospace Conference*, (Big Sky, MT), March 2000. Available from: `http://www.ssdl.gatech.edu/main/ssdl_paper_archive/IEEE-2000-P335E.pdf`.

[70] Keane, A. J. and Nair, P. B., "Problem Solving Environments in Aerospace Design," *Advances in Engineering Software*, vol. 32, pp. 477–487, 2001.

[71] Korsch, H. J. and Jodl, H.-J., *Chaos: A Program Collection for the PC*. Berlin: Springer-Verlag, 1994.

[72] Thom, R., *Stabilité Structurelle et Morphogenèse*. Reading: Benjamin, 1972.

[73] Castrigiano, D. P. L. and Hayes, S. A., *Catastrophe Theory*. Reading: Addison-Wesley, 1993.

[74] Rosser, J. B., *From Catastrophe to Chaos: A General Theory of Economic Discontinuities*, vol. 1. Boston: Kluwer Academic Publishers, 2 ed., 2000.

[75] Smith, T. S., "Catastrophes in Interaction: A Note on Arousal-Dependent Discontinuities in Attachment Behavior," *Social Psychology Quarterly*, vol. 57, pp. 274–282, September 1994.

[76] Cobb, L. and Zacks, S., "Applications of Catastrophe Theory for Statistical Modeling in the Biosciences," *Journal of the American Statistical Association*, vol. 80, pp. 793–802, December 1985.

[77] Alexander, R. M., *Optima for Animals*, ch. 3, pp. 45–57. Princeton: Princeton University Press, 1996.

[78] Baack, D. and Cullen, J. B., "Decentralization in Growth and Decline: A Catastrophe Theory Approach," *Behavioral Science*, vol. 39, pp. 213–229, July 1994.

[79] Svyantek, D. J. and DeShon, R. P., "The Illusion of Certainty: A Catastrophe Model of Decision Framing," *Current Psychology*, vol. 10, pp. 199–212, Fall 1991.

[80] Hines, R., Marsh, D., and Duffy, J., "Catastrophe Analysis of the Planar Two-Spring Mechanism," *The International Journal of Robotics Research*, vol. 17, pp. 89–101, January 1998.

[81] Broer, H. W., Hoveijn, I., van Noort, M., and Vegter, G., "The Inverted Pendulum: A Singularity Theory Approach," *Journal of Differential Equations*, vol. 157, pp. 120–149, 1999. Available from: `http://www.idealibrary.com`.

[82] Shanthini, W. and Nandakumar, K., "Bifurcation Phenomena of Generalized Newtonian Fluids in Curved Rectangular Ducts," *Journal of Non-Newtonian Fluid Mechanics*, vol. 22, pp. 35–60, 1986.

[83] Goldshtik, M. and Hussain, F., "Analysis of Inviscid Vortex Breakdown in a Semi-infinite Pipe," *Fluid Dynamics Research*, vol. 23, pp. 189–234, 1998.

[84] Chai, L. and Shoji, M., "Boiling Curves - Bifurcation and Catastrophe," *International Journal of Heat and Mass Transfer*, vol. 44, pp. 4175–4179, 2001. Available from: `http://www.elsevier.com/locate/ijhmt`.

[85] Murphy, J. W., "Catastrophe Theory: Implications for Probability," *American Journal of Economics and Sociology*, vol. 50, pp. 143–148, April 1991.

[86] Ashby, M. F., *Materials Selection in Mechanical Design*. Oxford: Pergamon Press, first ed., 1992.

[87] Johnson, K. W., Langdon, P., and Ashby, M. F., "Grouping Materials and Processes for the Designer: an Application of Cluster Analysis," *Materials and Design*, vol. 23, pp. 1–10, 2002.

[88] Maine, E. M. A. and Ashby, M. F., "An Investment Methodology for Materials," *Materials and Design*, vol. 23, pp. 279–306, 2002.

[89] Perloff, J. M., *Microeconomics*. Reading: Addison Wesley, 1999.

[90] Park, K.-W. and Grierson, D. E., "Pareto-Optimal Conceptual Design of the Structural Layout of Buildings Using a Multicriteria Genetic Algorithm," *Computer-aided Civil and Infrastructure Engineering*, vol. 14, pp. 163–170, 1999.

[91] Zitzler, E. and Thiele, L., "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 257–271, November 1999.

[92] Knowles, J. D. and Corne, D. W., "Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy," *Evolutionary Computation*, vol. 8, no. 2, pp. 149–172, 2000.

[93] Kumar, R. and Rockett, P., "Improved Sampling of the Pareto-Front in Multiobjective Genetic Optimizations by Steady-State Evolution: A Pareto Converging Genetic Algorithm," *Evolutionary Computation*, vol. 10, pp. 283–314, Fall 20052.

[94] Ismail-Yahaya, A. and Messac, A., "Effective Generation of the Pareto Frontier: The Normalized Constraint Method," in *43rd AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Meterial Conference*, (Denver, CO), April 2002.

[95] Roth, B., Graham, M., Mavris, D., and Macsotai, N., "Adaptive Selection of Pareto Optimal Engine Technology Solution Sets," in *Presented at the 2002 ICAS Conference*, (Toronto, Ontario), 2002. Available from: `http://www.asdl.gatech.edu/publications/pdf/2002/icas2002_paper_594.pdf`.

[96] Mattson, C. A. and Messac, A., "Concept Selection Using s-Pareto Frontiers," *AIAA Journal*, vol. 41, pp. 1190–1198, June 2003.

[97] Mattson, C. A. and Messac, A., "Pareto Frontier Based Concept Selection Under Uncertainty, with Visualization," *Optimization and Engineering*, 2003.

[98] Arnol'd, V. I., *Catastrophe Theory*. Berlin: Springer-Verlag, 3 ed., 1992.

[99] Hale, F. J., *Introduction to Aircraft Performance, Selection, and Design*. New York: John Wiley & Sons, 1984.

[100] Wang, T., *Global Optimization for Constrained Nonlinear Programming*. PhD thesis, University of Illinois at Urbana-Champaign, 2001.

[101] Powell, M. J. D., "An efficient method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives," *The Computer Journal*, vol. 7, pp. 155–162, July 1964.

[102] Fletcher, R. and Reeves, C. M., "Function Minimization by Conjugate Gradients," *The Computer Journal*, vol. 7, pp. 149–154, July 1964.

[103] Nelder, J. A. and Mead, R., "A Simplex Method for Function Minimization," *The Computer Journal*, vol. 7, pp. 308–313, January 1965.

[104] Luenberger, D. G., *Linear and Nonlinear Programming*. Reading: Addison-Wesley, second ed., 1984.

[105] Zoutendijk, G., *Method of Feasible Directions: A Study in Linear and Non-Linear Programming*. Amsterdam: Elsevier, 1960.

[106] Morey, C., Scales, J., and Van Vleck, E. S., "A Feedback Algorithm for Determining Search Parameters for Monte Carlo Optimization," *Journal of Computational Physics*, vol. 146, pp. 263–281, 1998.

[107] Hollingsworth, P. M. and Mavris, D. N., "Determination of Revolutionary Requirements Boundaries for a High-speed, Airbreathing Propulsion System," in *2nd Aerospace Technologies Integration and Operation Conference*, (Los Angeles, CA), American Institute of Aeronautics and Astronautics, 2002.

[108] Coello Coello, C. A., "A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques," *Knowledge and Information Systems*, vol. 1, no. 3, pp. 129–156, 1999. Available from: `citeseer.nj.nec.com/coello98comprehensive.html`.

[109] Coello Coello, C. A., "An Updated Survey of Evolutionary Multiobjective Optimization Techniques: State of the Art and Future Trends," in *Proceedings of the Congress on Evolutionary Computation* (Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., eds.), vol. 1, (Mayflower Hotel, Washington D.C., USA), pp. 3–13, IEEE Press, 6-9 1999. Available from: `citeseer.nj.nec.com/coellocoello99updated.html`.

[110] Zitzler, E., Deb, K., and Thiele, L., "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," *Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, 2000.

[111] Deb, K., Pratap, A., Agarwal, S., and Mayarivan, T., "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, April 2002.

[112] "Test Problem Suite," November, 4 2003. Available from: http://www.tik.ee.ethz.ch/~zitzler/testdata.html/.

[113] Zitzler, E., Laumanns, M., and Thiele, L., "SPEA2: Improvng the Strength Pareto Evolutionary Algorithm," Tech. Rep. TIK-Report 103, Swiss Federal Institute of Technolog (ETH) Zürich, ETH Zentrum, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001. Available from: http://citeseer.nj.nec.com/rd/0%2C468146%2C1%2C0.25%2CDownload/ftp%3AqSqqSqftp.tik.ee.ethz.chqSqpubqSqpeopleqSqlaumannsqSqZLT2001a.pdf.

[114] Hollingsworth, P. and Mavris, D. N., "A Technique for Use of Gaussian Processes in Advanced Meta-modeling," in *SAE Aerospace Congress and Exhibition*, (Montreal, Canada), Society of Automotive Engineers, September 2003.

[115] "An Analysis of US Army Helicopter Programs," tech. rep., Congrssional Budget Office, 1995. Available from: http://www.cbo.gov/showdoc.cfm?index=12&sequence=2 [cited November 26, 2003].

[116] Keys, C. N., Schmidt, A. H., and Donnelly, J. P., "Application of Stealth Technology to the LHX Concept Definition," Tech. Rep. DAAK50-81-G-0003, Boeing Vertol Company, Philadelphia, PA, 1983.

[117] Zellner, R., "RAH-66 Comanche Photo," November 2003. Available from: http://www.boeing.com/rotorcraft/military/rah66/images/RAH66_web_shot_1.htm [cited November 26, 2003].

[118] Meier, W. H., "Application of the Advancing Blade Concept to the LHX Concept Development," Tech. Rep. SER-510096, Sikorsky Aircraft, E, 1983.

[119] "Boeing Sikorsky RAH-66 comanche technical specifications.". Available from: http://www.boeing.com/rotorcraft/military/rah66/rah66tech.htm [cited December 4, 2003].

[120] Pike, J., "RAH-66 Comanche," November 2003. Available from: http://www.globalsecurity.org/military/systems/aircraft/rah-66.htm [cited November 26, 2003].

[121] "LHTEC: Ratings chart.". Available from: http://www.lhtec.com/ratechrt.html [cited December 8, 2003].

[122] Martin, T., "Asymptotic," 2002. Available from: http://www.asymptotic.co.uk/ [cited February 10, 2003].

[123] Gilmore, R., *Catastrophe Theory for Scientists and Engineers*. New York: Dover, 1993.

[124] Poston, T. and Stewart, I., *Catastrophe Theory and its Applications*. Mineola: Dover, 1978.

[125] Arnol'd, V. I., *Catastrophe Theory*, ch. 4, pp. 10–13. Berlin: Springer-Verlag, 1986.

[126] Mahoney, J. J., *Inlets for Supersonic Missiles*, ch. 11, p. 139. Washington: American Institute of Aeronautics and Astronautics, 1990.

[127] *RAMSCRAM Users Manual*. Cleveland, Ohio.

[128] Joy, D. P. and Simonds, R. M., "The RF Graphical Method of Parametric Analysis for the Development of Optimum Preliminary Design Aircraft," paper, Advanced Research Division of Hiller Helicopters, February 1956.

[129] Simonds, R., "A Generalized Graphical Method of Minimum Gross Weight Estimation," in *The National conference of the Society of Aeronautical Weight Engineers*, (San Diego, CA), Society of Aeronautical Weight Engineers, Inc., May 1956.

[130] Roskam, J., *Airplane Design*, vol. 1. Lawrence: Design Analysis and Research Corporation, second ed., 1989.

[131] Raymer, D. P., *Aircraft Design: A Conceptual Approach*. Reston: American Institute of Aeronautics and Astronautics, 1999.

[132] Mattingly, J. D., Heiser, W. H., and Daley, D. H., *Aircraft Engine Design*. New York: American Institute of Aeronautics and Astronautics, 1987.

[133] Schrage, D. P., Mavris, D. N., and Wasikowski, M., "GTPDP - A Rotary Wing Aircraft Preliminary Design and Performance Estimation Program Including Optimization and Cost," in *Vertical Lift Aircraft Design Conference*, (San Francisco, CA), January 1990.

VITA

Peter Hollingsworth was born in Enid Oklahoma in 1976. Subsequently he lived in Williamsburg, Virginia and Cincinnati, Ohio. After graduation from Mariemont High School in Cincinnati in 1994, he moved to Atlanta to pursue studies in Aerospace Engineering at the Georgia Institute of Technology. He graduated from Georgia Tech with a Bachelors of Aerospace Engineering in 1999 and a Masters of Science in Aerospace Engineering in 2000. Subsequently he has been pursuing his Ph.D. studies in advanced systems design methodologies at the Aerospace Systems Design Laboratory at Georgia Tech. His primary research focus has been in developing methods to understand the effect of system level requirements on complex system behavior and properties, and technology policy.